Security Assurance Multi-Agent based Monitoring System for Critical Infrastructures

Abstract

Open systems as telecommunications infrastructures are massively distributed. They are composed of highly connected sets of managed products. There is no general way to measure the confidence operators and end customers can have in the security of the infrastructure, in end-to-end security services and in the security of end-to-end services above those architectures.

This project aims to define a security framework to measure, document and maintain the security assurance level of services based on telecommunication system.

The security framework will provide guidelines and methods, as well as software applications, to assess the overall confidence that can be obtained. The framework will be based on a specific middleware, developed using technologies such as mobile agents, to collect information within infrastructures in a non-disturbing and non-intrusive way. Information will be collected by applications such as vulnerability automatic research engine, protocol security analyzer, and will be include configurations management information (linked to databases of certified configuration) to automate security resting.

The project will deliver a system security cockpit, built using the security framework, to help equipment manufactures, networks architects, and operators reach and maintain a certain level of security assurance. The cockpit could show certified components/configurations within the architecture, identify assurance domains and be an assistant to remotely and automatically launch specific tests on equipment to augment/verify the security quality assurance level (automated, remote non-intrusive audits), etc. Applications of this security cockpit will address interaction and integration into Network Management Systems (NMS) and interfaces to Operation Support Systems (OSS).

One principal goal of this work is the development of a framework, which aims to integrate and coordinate existing security tools and to ensure the global security assurance level of services based on telecommunications systems, but also to maintain this assurance level through permanent monitoring (Security Cockpit). The main focus of this paper is to develop a prototype of this framework.

TABLE OF CONTENTS

Security Assurance Multi-Agent based Monitoring System for Critical Infrastructures 1
Table of Contents 4
List of Figures 7
1 Introduction and problem statement 9
2 Requirements Analysis 11
2.1 Description of the SESAME objectives 11
2.2 Terminology 12
2.2.1 Assurance 12
2.2.2 Metric 14
2.2.3 Probe, Agent and Security Cockpit 16
2.3 User Requirements 16
2.3.1 Framework overview 16
2.3.2 BUGYO system 19
2.3.3 Probes 19
2.3.4 Compliance 20
2.3.5 Vulnerability 20
3 Architecture Design and Modelling 22
3.1 Architecture 22
3.1.1 Network topology 22
3.1.2 Use of a Multi-Agent System 22
3.1.3 Network Topology with BUGYO Agent Based Model 24
3.1.4 Agent Inheritance Model 26
3.1.5 Communication Flow and Formats 26
3.1.6 The Command file 33
3.1.7 Agent Diagram 36
3.1.8 Agents Organization 37
3.2 Agent Design Pattern 39
3.3 BUGYO Agent 41
3.3.1 Description 41
3.3.2 Methods implemented in the BugyoAgent class 41
3.3.3 Behaviours implemented in the BugyoAgent class 42
3.4 BUGYO Interface 43
3.4.1 Description 43
3.4.2 Methods Implemented in the BugyoInterface class 43

3.5 Appliance Agent 43
3.5.1 description 43
3.5.2 Methods Implemented in the ApplianceAgent class 43
3.5.3 Behaviours implemented in the ApplianceAgent class 44
3.6 Server Agent 44
3.6.1 Description 44
3.6.2 Methods contained in the ServerAgent class 44
3.6.3 Behaviours implemented in the ServerAgent class 44
4 Technological Solutions Identification 46
4.1 Multi Agent System (MAS) 46
4.1.1 Introduction 46
4.1.2 MAS Overview 46
4.1.3 Different MAS platforms 48
4.1.4 Conclusion 61
4.2 Probes 63
4.2.1 OVAL 63
4.2.2 Nagios 72
5 Implementation 74
5.1 BADK 74
5.2 ApplianceAgent and ServerAgent 74
5.3 Specifics Agents 74
5.3.1 Template – Probe Integration Specification 74
5.3.2 OVAL agent 76
5.3.3 Nagios agent 77
5.3.4 SSL Agent 77
5.4 Graphic User Interface 79
6 Testing 80
6.1 Testing Architecture 80
6.2 BUGYO Test Platform 80
6.3 Web server with SSL 81
7 Conclusion 83
8 References 85
Appendices 86
8.1 Detailed description of implemented methods 86
8.1.1 BugyoAgent 86
8.1.2 BugyoInterface 90

8.1.3 ApplianceAgent *90*
8.1.4 ServerAgent *91*
8.2 Defining own definition file 92
8.3 Using OVAL Definition Interpreter 97
8.4 Install and run Nagios 101
8.5 Install XAMPP 1.5.2 on SUSE Linux 10.1 104
Glossary 106

LIST OF FIGURES

Figure 1 – Internship view 11
Figure 2 - Security Assurance 13
Figure 3 – Metric representation 14
Figure 4 - High-level view of the framework 18
Figure 6 - Network Topology without BUGYO 22
Figure 7 – Network Topology with BUGYO Agent Based Model 25
Figure 8 – Agent Inheritance 26
Figure 9 - Message Flows 27
Figure 10 - XML data sent to/by the appliance 29
Figure 11 - XML Schema for Server-Appliance format 32
Figure 12 – XML data sent to/by the agent 33
Figure 13 - Content of the .xml file 34
Figure 14 - XML Schema corresponding to command file (XML) 35
Figure 15 - Agent Diagram 36
Figure 16 – Agents management 37
Figure 17 - Class diagram 41
Figure 18 - JADE Agent Communication Channel 49
Figure 19 - JADE Platform Architecture 49
Figure 20 – JADE Agent Deployment 51
Figure 21 - JADE Platform Security 53
Figure 22 - How to use OVAL 70
Figure 23 - Template - Probe Integration Specifications 76
Figure 24 – Command file for OVAL Definition Interpreter probe 77
Figure 25- Command file for SSL agent 78
Figure 26 - Graphic User Interface 79
Figure 27 – Testing Architecture 80
Figure 28 - Prototype Architecture 81
Figure 29 - Windows self-extracting archive 99
Figure 30 - Execution of the Interpreter 100

Figure 31 - OVAL results..... 101

1 INTRODUCTION AND PROBLEM STATEMENT

Currently, there is no framework to assess and manage the assurance level of a system such as Telecommunications and services infrastructures. Current methodologies such as ISO 17799 are not applicable to large deployed system and focus more on organizations and small system.

Some tools exist to audit and test telecom/protocol base system, detecting weakness such as open ports, but not take into account combinations and composition of different elements. These issues are addressed within BUGYO [1] project where I work for during my internship.

The main focus of the BUGYO project is to build a framework providing and integrating means to measure the security assurance of a telecommunication infrastructure, as well as the provided services above it. The project addresses security, as a full system approach, and the major expected result will be a system security assurance framework including methodologies, best practices, tools and certification cockpit.

The aim is to develop a framework of security to measure, document and maintain the security assurance level of telecommunication systems. More particularly, the two associated objectives are:

1. Defining a security framework: this one provides methodology and the good practices to be implemented in order to ensure the sufficient security level and to reach the required trust.
2. The development of monitoring security tools, which will come in support from methodology suggested, and which will make sure that each intermediary (salesmen of equipment, architects of telecommunications networks, communication's operators, service's suppliers, etc.) contribute to guarantee the security assurance level wished. These tools are indicated in the project under the name of "Security Cockpit".

Security Assurance has become a profitable business for consulting companies. Since recently dedicated security assurance products start appearing on the market [2][14] [15]

but these commercial offers both on consulting firm offers and commercial tools demonstrates that security assurance is an opening and looking forward market opportunities.

We can distinguish three business cases for BUGYO project, which reflects three types of results.

1. Monitoring

   The GUI of the cockpit allows fallow-up the security assurance of several services. This can be integrated in a NOC (Network Operation Center) on the same basis as the service quality monitoring on larges telecom services. It can also been integrated in a SOC (Security Operation Center) in order to complete the security monitoring.

2. Alarm

   BUGYO can emit alarms on the metric or the service level. These alarms will be sent to the network administrator. The big innovation of these alarms is that one alarm contains a criticality level; this has the advantages that it indicates the administrators which alarm has to be treat in priority.

3. Audit reports

   Each change of the security assurance level will be logged (saved). This permit to get the history of the changes of the security assurance level.

2 REQUIREMENTS ANALYSIS

2.1 Description of the SESAME objectives

One principal goal of this project is the development of a framework, which aims to integrate and coordinate existing security tools and to ensure the global security assurance level of services based on telecommunications systems, but also to maintain this assurance level through permanent monitoring (Security Cockpit).

The goal of this internship is not to develop the complete framework described above, but to develop a prototype of this framework. To have a better view of the internship objective, we can look at the fallowing figure, which identify the two main points of the prototype: Probes&Tools and Methodology.

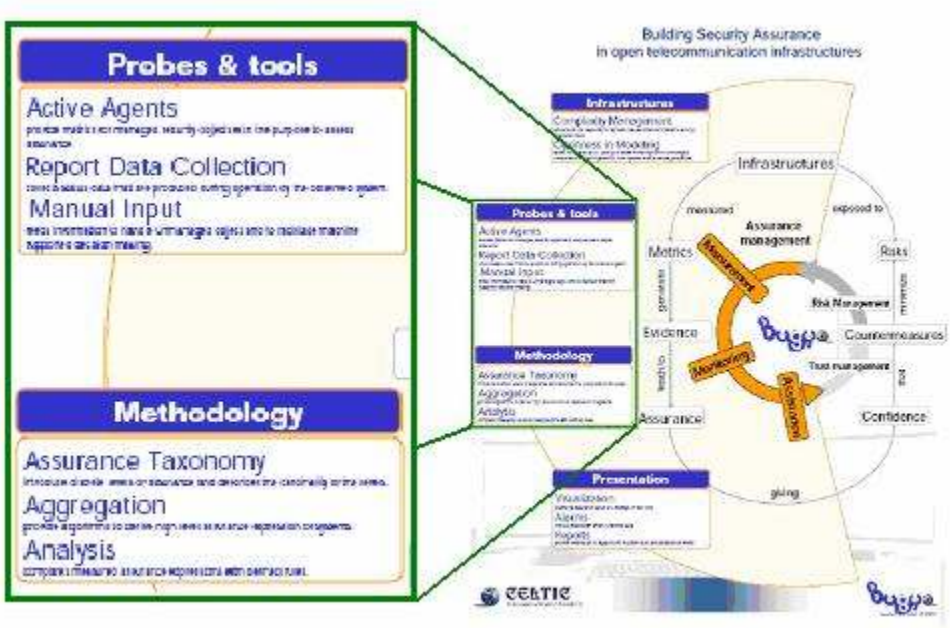

Figure 1 – Internship view

The actual agent-based prototype, aims in a first step to show how we can collect distributed data about network elements of an infrastructure using existing security tools. At this level we won't take in account the aggregation and the presentation of the collected data. Then it would include a GUI for the collected data presentation.

The prototype belongs to the WP3 and takes in account WP1 and WP2. The results provided by this prototype could be exploited in the WP4, which allows using the cockpit to represent the received data. Three screens represent this cockpit.

With this prototype we want to show the assurance level of a simple infrastructure. At the end, we will demonstrate if changing some parameters of this infrastructure, then the assurance level changes.

My internship can easily be divided into four parts. Initially, I have read all documents about the project in order to understand the project's goal, where will it be used, who will use it, and what are the internship objectives.

Then, a long technological survey fallows: in a first step surveys about multi-agent systems (MAS) in general and their appropriate usage under this project, then more precisely on the existing platforms for finally choosing the most appropriated to BUGYO. In a second step surveys about existing security tools, which could be used in BUGYO project, with at the end a small prototype to demonstrate how to use a MAS and an existing security tool.

2.2 Terminology

We need to specify some vocabulary related to the project that we will be using along this document.

2.2.1 ASSURANCE

The following Figure (extracted and modified from ISO 15408-1) shows what security assurance is. The BUGYO project will address the colored boxes and associated relationships with a special focus on the assurance techniques part. **Assurance is a measure of the confidence** that can be to given to end users that the security countermeasures deployed in a telecommunications system will minimize risks that can occur to their assets. Assets can be very different depending on services.

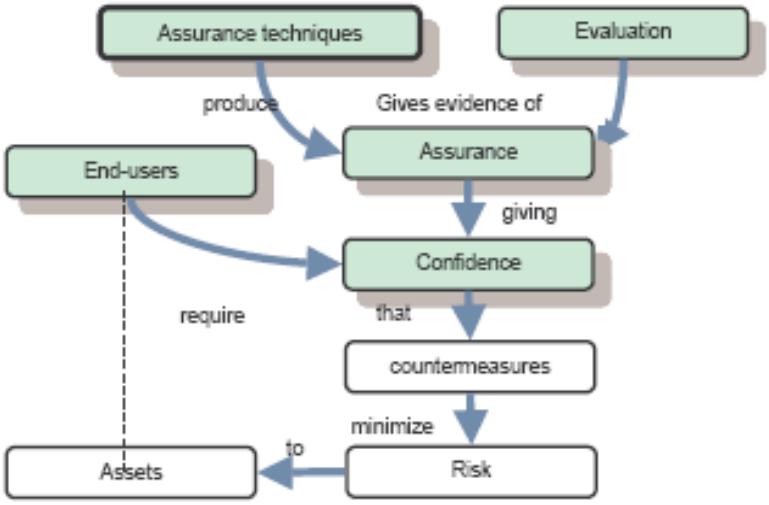

Figure 2 - Security Assurance

2.2.2 METRIC

For each security function (that performs a counter-measure), a metric is evaluated. A metric is a set of measures belonging to one of the three categories:

- Availability,
- Compliance
- and Vulnerability.

There are two levels of aggregation to obtain the assurance metric of a security function. The first one aims to aggregate measures in meta-measures (availability, compliance and vulnerability). The second one aims to aggregate these meta-measures. The aggregation functions might be generic or specific to a metric

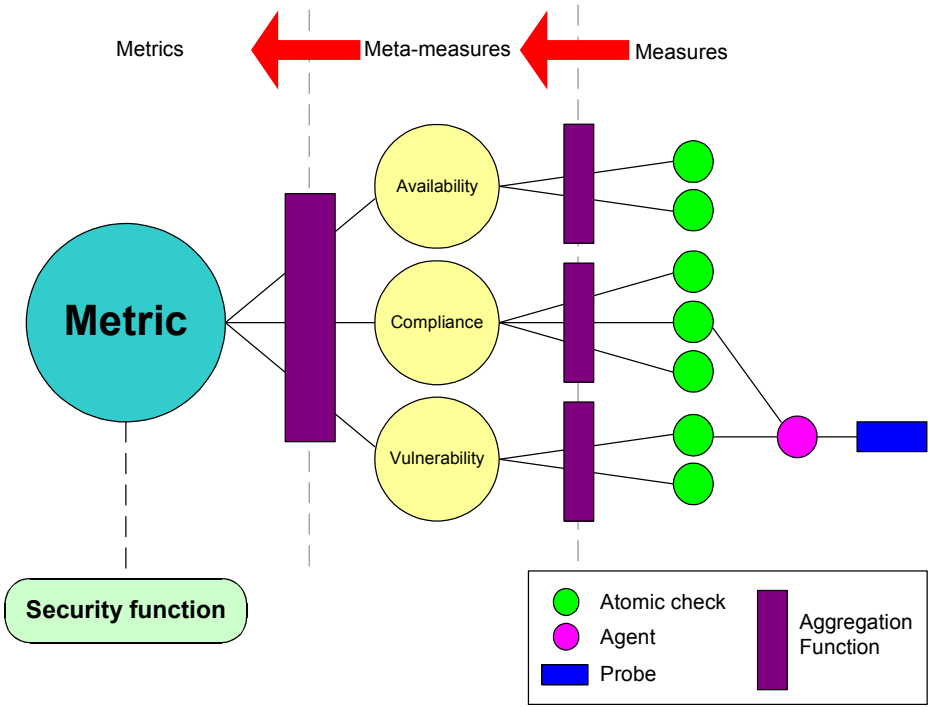

Figure 3 – Metric representation

Metric Assurance Level (MAL) is a stamp that indicates a normalized level at which a metric is certified/accredited for the security assurance evaluation on an atomic security function.

Security Function Assurance Rate (SFAR) is the value that is processed using weight.

Security Function is the same as definition in the Common Criteria (ISO 15408.)

The Common Criteria (CC) [4] is an international standard (ISO/IEC 15408) for computer security. Unlike standards such as FIPS 140, Common Criteria does not provide a list of security requirements to be satisfied. Instead, it describes a framework in which users can specify their security requirements, developers can make claims about the security attributes of their products, and evaluators can determine if products actually meet their claims. In other words, Common Criteria provides assurance that the process of specifying, developing, and evaluating a computer security product has been conducted in a rigorous manner.

2.2.3 PROBE, AGENT AND SECURITY COCKPIT

Probe is a tool that gathers information necessary to measure the assurance level at the local node or at a Network Entity level.

In the telecom world an agent is an interface between probes and the Security Cockpit.

The Security cockpit is a system that performs measurements of the security assurance level of the whole system.

2.3 User Requirements

The technological innovation is mainly in how to realize security assurance assessment on an operational telecommunications system, by launching tests and collecting results in a distributed way. Innovation may involve technology such as mobile agents technology middleware as a vector of test and means to gather results. Other point will be a system security cockpit that will analyze all the result and provide a synthesis assurance level similar to product Common Criteria EALs.

According to Raymond James & Associates, Inc., to address security issues successfully, three functions must be incorporated into the process.

1. Security assessment and audits,
2. Network security architecture, implementation, and integration
3. Ongoing monitoring and management of the infrastructure

2.3.1 FRAMEWORK OVERVIEW

The fallowing figure describes a high level view of the framework. It is built in three main areas:

1. The telecom operations aspect addresses specificities from a telecommunication infrastructure, and provides a model of the underlying infrastructure.
2. An integration framework provides the necessary means to integrate and coordinate different security tools as well as different security abstraction

models in order to define different security profiles for the assessment of the security assurance of a determined service.

3. The System Security Cockpit represents the interface for the operator or the service provider to perform necessary operations to obtain and maintain a security assurance level for a specific service.

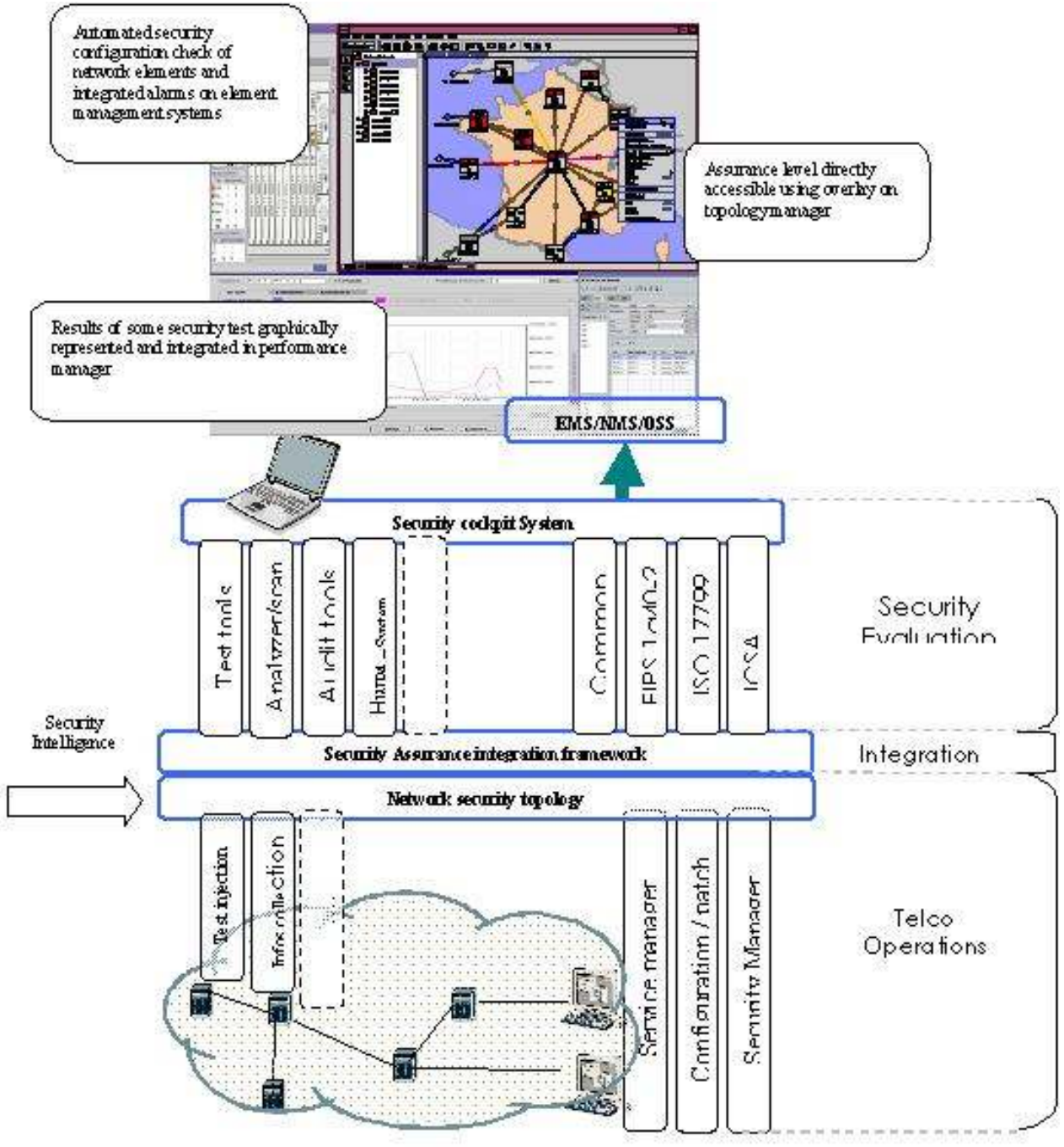

Figure 4 - High-level view of the framework

2.3.2 BUGYO SYSTEM

The BUGYO system is made up of:

- A GUI also called "security cockpit". It is in charge of the assurance level display.
- A server, which is in charge of the assurance level evaluation.
- Agents in charge of the collection of assurance metrics.
- Interfaces also called Appliance, which are in charge of the communication between the server and the agents.
- Probes, which are tools that perform real measurement.

2.3.3 PROBES

Probes can be COTS or BUGYO-specific tools.

- In the first case, probes receive requests from or send data to BUGYO agents that are in charge of the communication with the security cockpit.
- In the second case, probes directly communicate with the security cockpit. They integrate natively the agent functionality.

There are three kinds of probes:

- Monitoring probes.
  This kind of probes will emit information to the system without being requested.
- Audit probes.
  This kind of probes will responds to the requests of the system.
- Monitoring and audit probes.
  This kind of probes will respond to the requests to the system or emit information to system without being requested.

Probes can be located in or out of the machine hosting the NE. They can have three different forms depending on their location: local/remote, active/passive, work on demand or permanently.

- Remote active measurement:
  - active scans (e.g. nmap, Nessus)

- Remote passive measurement.
  - a new machine is detected (PADS, Arpwatch)
- Local measurement: the probe (command, specific piece of code, COTS) is located on the NE.
  - Check for integrity loss of any security device configuration file (Samhain, Tripwire).

2.3.4 COMPLIANCE

Due to deployment or security constraints, there are several ways to check the compliance of a counter-measure with the security policy:

- By a direct measurement.
  It consists in getting the counter-measure parameters and in analyzing them. This is the best way to check the compliance.
- By an indirect measurement.
  It is not always possible to access the counter-measure parameters. The compliance is then deduced from indirect measurements.
  - A priori indirect measurement. Such a check is typically performed during an audit.
  - A posteriori indirect measurement. Such a check is typically a security alarm that points out a flaw in the applied security policy.

Functional specifications:

- Check of the authentication policy
- Check of the security rules of a firewall
- Check of the integrity control

2.3.5 VULNERABILITY

The presence of vulnerabilities on a security function has to be taken into account for the assurance evaluation of this security function.

Check of the security function vulnerability

- The system must be able to launch scans for known vulnerabilities on demand or periodically.
- The system must be able to collect and display the known vulnerabilities for every security function.
- The system must be able to collect and display the exploitable vulnerabilities for every security function.

Patch management

- The system must be able to compare the patch level of a security function with a patch management tool database.

3 ARCHITECTURE DESIGN AND MODELLING

Initially, this chapter presents an agent-based architecture and the reason why choosing a multi-agent system. The chapter also describes the different agent contained in the infrastructure, their heritance and organization model and the format for messages exchanges between each other. And finally, a detailed description of the methods and behaviours implemented for each agent.

3.1 Architecture

3.1.1 NETWORK TOPOLOGY

The basic network topology without BUGYO is illustrated below:

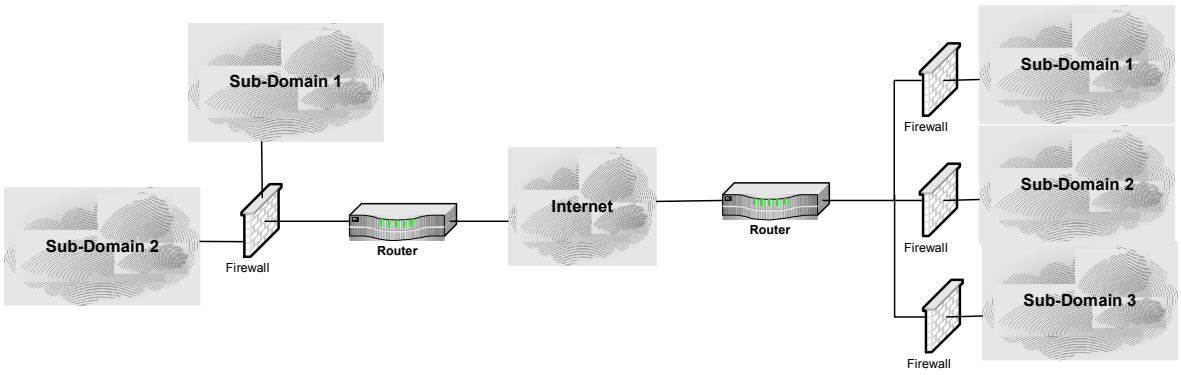

Figure 5 - Network Topology without BUGYO

3.1.2 USE OF A MULTI-AGENT SYSTEM

There were many technologies, which could be used to develop the prototype. However we choose to use a multi-agent system for the fallowing reasons.

The user requirements talks about using an agent technology, but in telecom world agents doesn't have the same meaning as in the IT's world. In telecom domain an agent is a simple identity, which would transform probes results in BUGYO results.

The fact of having an infrastructure massively distributed is one reason of choosing a multi-agent system. Such technology can easily be deployed in a distributed system.

A Multi-Agent platform offers many advantages, which could be profitable for BUGYO framework: Autonomy, Intelligence, Mobility, Robustness, Scalability, Flexibility, API already presents and communication language and architecture already defined. After the necessary measurements have been performed on a NE, an agent could move to another NE to perform the same type of measurements.

Further, using agents in distributed computing offers the possibility to support heterogeneous systems, adaptability and loose coupling.

Security is another advantage offered by some Multi-Agent platforms. This permits to exchange secure messages and data between several agents.

3.1.3 NETWORK TOPOLOGY WITH BUGYO AGENT BASED MODEL

In the proposed model (Figure 14), several type of agent are deployed controlled by the MAS Control and offering a distributed architecture. Each MAS agent requires an MAS Runtime Environment.

Server Agent

The MAS platform is controlled by the MAS Control installed into BUGYO Server. An agent, called Server Agent is in charge of sending and collecting information to the appliance agents. This agent is also in charge of aggregating information received from other agents. Server Agent could interact with a database for persistency.

Appliance Agent

Each appliance is in charge of only collecting the measures. There is one appliance per sub-domain. Each appliance can handle one to several agents.

Probe Agent

Each agent is dedicated to a probe. Therefore each agent can monitor one probe for scalability and implementation facilities. The probes could be located within the appliance server or outside of the appliance depending on the deployment of the agent. Moreover, a human operator could also be considered as an agent for manual inputs. One probe could provide several measures or controls. Also one agent can perform several measures.

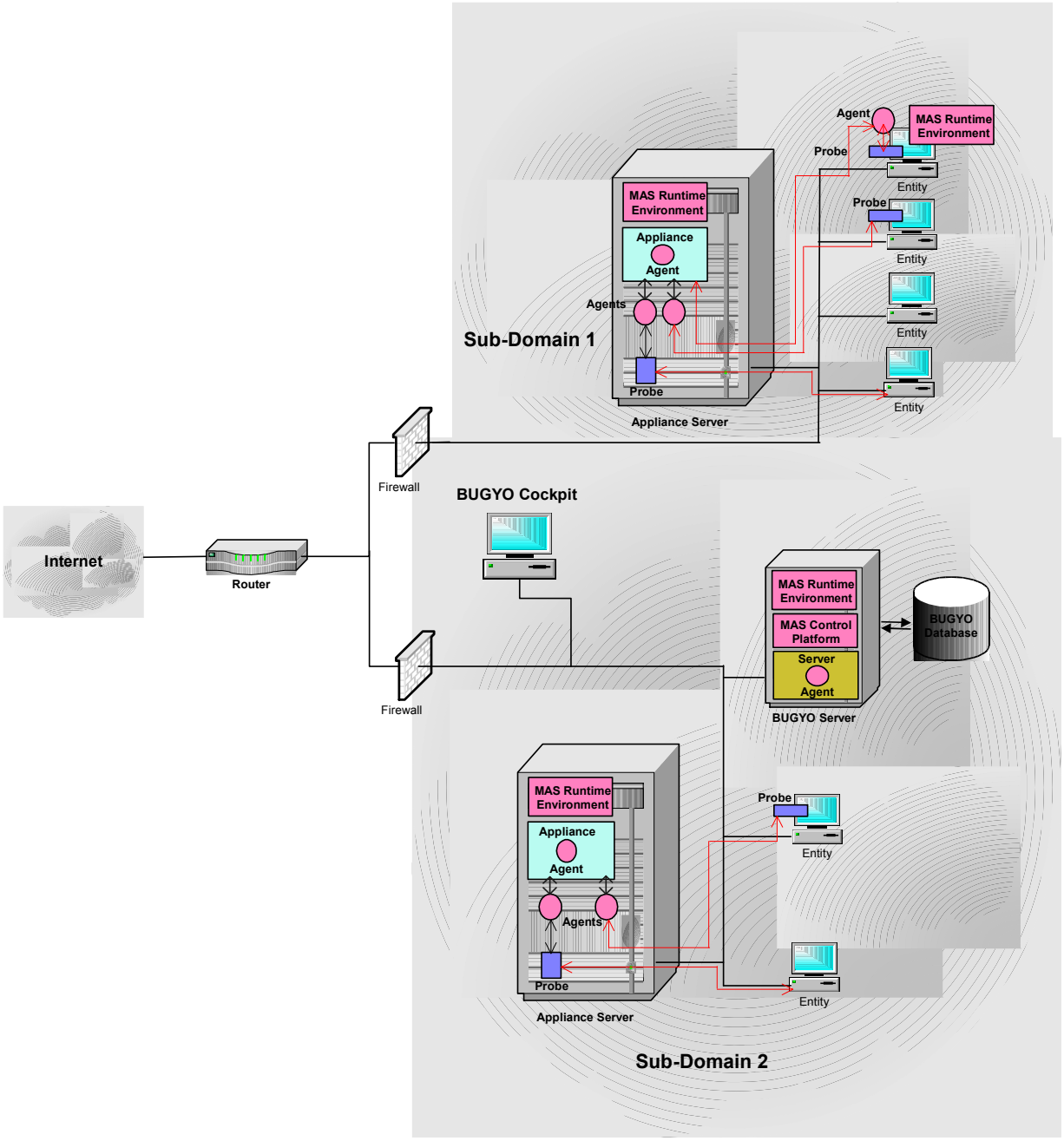

Figure 6 – Network Topology with BUGYO Agent Based Model

3.1.4 AGENT INHERITANCE MODEL

The Server Agent, Appliance Agent and Generic Probe Agent inherit from Agent provided by the MAS Platform, which would be used (Figure 15). These agents will then implement specific method regarding their functionalities as described in the last section.

Generic Probe Agent implements generic methods for probe monitoring. For each probe, specific Probe Agent implements specific methods dedicated to the control of the probe.

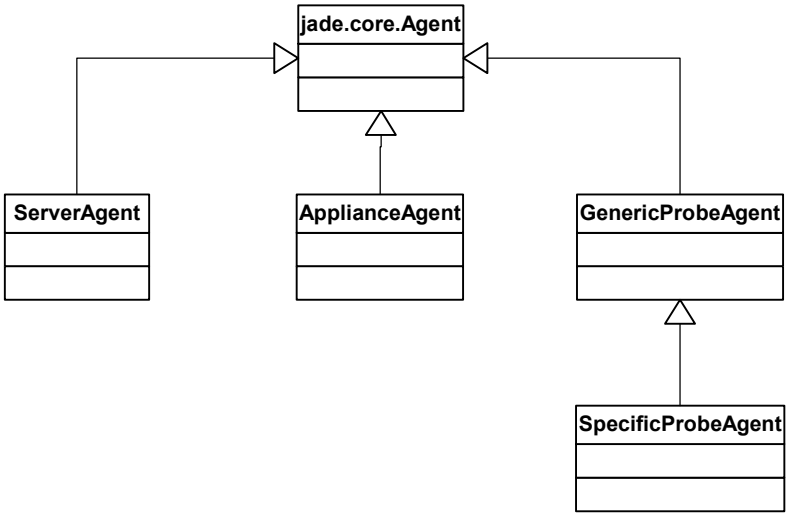

Figure 7 – Agent Inheritance

With such agent inheritance, developers are able to develop their own specific agents without having any knowledge of multi-agent system (or jade more precisely).

3.1.5 COMMUNICATION FLOW AND FORMATS

3.1.5.1 Message Flow

Agents receive messages and translate them into understandable format for each probe. The appliance agent is in charge of converting messages from each agent to normalized BUGYO format and from normalized BUGYO format to understandable format for each agent controlling the probes.

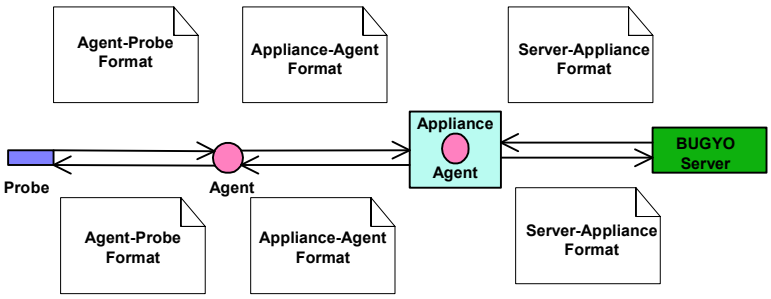

Figure 8 - Message Flows

3.1.5.2 Message Formats

As presented in figure, we will detail the specification of all the messages in the fallowing subsections.

Server-Appliance Format

The server sends to the appliance and receives from the appliance the following XML based formatted message:

```
<metrics>
        <metric id= Metric_1>
                <measurements>
                        <measurement id= Measurement_1>
                                <target>X.X.X.X</target>
                                <value>xxx</value>
                                <logs>logs_X</logs>
                        </measurement>
                        <measurement id = Measurement_2>
                                <target>Y.Y.Y.Y</target>
                                <value>yyy</value>
                                <logs>logs_Y</logs>
                        </measurement>
                        <measurement id =Measurement_3>
                                <target>Z.Z.Z.Z</target>
                                <value>zzz</value>
                                <logs>logs_Z</logs>
                        </measurement>
                        …
                        <measurement id= Measurement_k >
                                <target>K.K.K.K</target>
                                <value>kkk</value>
                                <logs>logs_K</logs>
                        </measurement>
                </measurements>
        </metric>
        <metric id= Metric_2>
                <measurements>
                …
                </measurements>
        </metric>
        …
        <metric id=Metric_i >
                <measurements>
                …
                </measurements>
        </metric>
</metrics>
```

Figure 9 - XML data sent to/by the appliance

When the server sends the message to the appliance, the values of different measurements are empty. When the appliance sends the message back, all the measurements are performed and the values of the measurements are filled.

XML-Schema corresponding to Server-Appliance XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="metrics">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="metric"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="metric">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="measurements"/>
            </xs:sequence>
            <xs:attribute name="id" use="required" type="xs:NCName"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="measurements">
        <xs:complexType>
            <xs:sequence>
            <xs:element maxOccurs="unbounded" ref="measurement"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="measurement">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="target"/>
                <xs:element ref="value"/>
                <xs:element ref="logs"/>
            </xs:sequence>
        <xs:attribute name="id" use="required" type="xs:integer"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="target" type="xs:NCName"/>
    <xs:element name="value" type="xs:NCName"/>
```

```
        <xs:element name="logs" type="xs:NCName"/>
</xs:schema>
```

Figure 10 - XML Schema for Server-Appliance format

Appliance-Agent Format

An agent can manage one or several probes. The appliance sends to the different agent the part of the message above, which concerns only each agent. As a consequence, each agent will receive (with no values) and will provide to the appliance (with values) such messages:

```
<measurements>
        <measurement id= Measurement_1>
                <target>X.X.X.X</target>
                <value>xxx</value>
                <logs>logs_X</logs>
        </measurement>
        <measurement id =Measurement_2>
                <target>Y.Y.Y.Y</target>
                <value>yyy</value>
                <logs>logs_Y</logs>
        </measurement>
</measurements>
```

Figure 11 – XML data sent to/by the agent

Agent-Probe Format

This format depends on what the probe needs as data format and on what the probe generates. The agent is in charge of handling this format and doing the transformation between the received messages by the appliance.

3.1.6 THE COMMAND FILE

During a probe's life cycle, some parameters could change. For example, some probe need authentication per login / password, or dictionary file. These kinds of information must not be directly integrated in the java code, but must be given as parameter to the agent. That's why a XML file containing these kinds of information must be created.

Each agent would have access to its own .xml file. The content of this file is described in the next figure.

```
<measurements>
        <measurement id="measurement_ID">
                <command dir="directory of
                command">command</command>
                <parameter id="parameter_ID" value="parameter_value"/>
        </measurement>
        …
        <measurement id="measurement_ID">
                <command dir="directory of
                command">command</command>
                <parameter id="parameter_ID" value="parameter_value"/>
        </measurement>
</measurements>
```

Figure 12 - Content of the .xml file

For each measurement's id (id), the specific command (command) containing parameters (parameters) has to be launched in the directory (dir). Parameters must be identified as @bygyo_XXX@ where XXX is the parameter's name. The reserved parameter @bugyo_target@ is representing the target's address.

XML schema corresponding to the XML file described above:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
	<xs:element name="measurements">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="measurement"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="measurement">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="command"/>
				<xs:element maxOccurs="unbounded" ref="parameter"/>
			</xs:sequence>
			<xs:attribute name="id" use="required" type="xs:integer"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="command">
		<xs:complexType mixed="true">
			<xs:attribute name="dir" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="parameter">
		<xs:complexType>
			<xs:attribute name="id" use="required" type="xs:NCName"/>
			<xs:attribute name="value" use="required" type="xs:NCName"/>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Figure 13 - XML Schema corresponding to command file (XML)

3.1.7 AGENT DIAGRAM

The agent diagram includes four types of elements:

Agent types represented by circles.

Humans: users that must interact with the system; they are represented by the UML actor symbol

Resources: external systems that must interact with the system; they are represented by rectangles

Acquaintances: represented by an arrow linking instances of the above elements, specifying that the linked elements will have to interact in some way while the system is in operation.

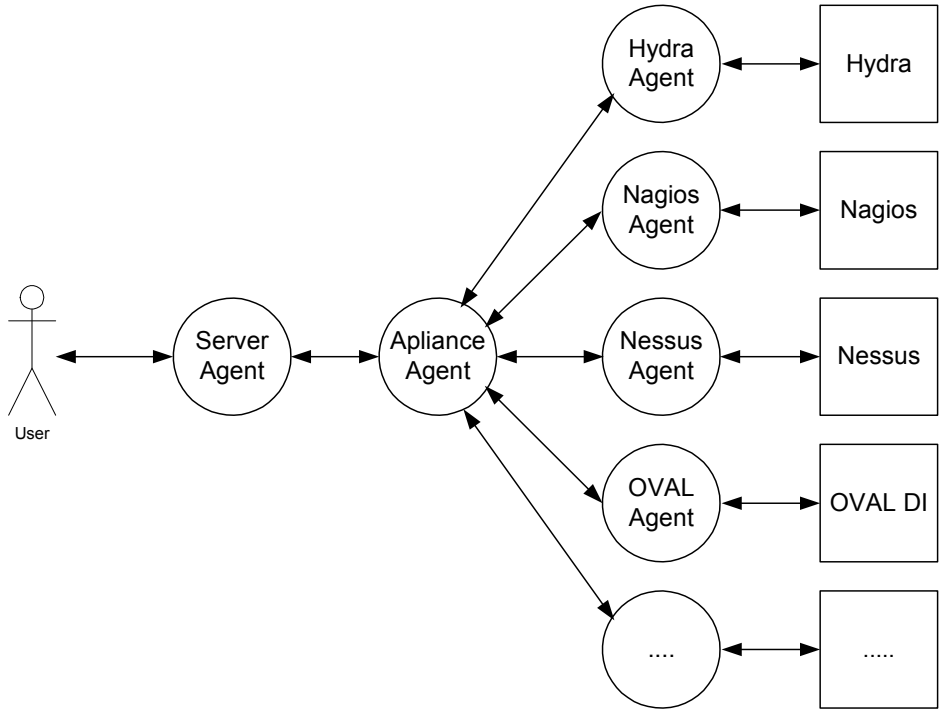

Figure 14 - Agent Diagram

3.1.8 AGENTS ORGANIZATION

One probe could provide several measures and one agent can perform several measures. Also one agent could provide several atomic checks. All specific probe agents are registered with their atomic checks (in MAS domain we talk about services) in a directory facilitator (Figure 23) and then published.

The appliance agent can ask to perform several atomic checks. It first requests to the directory facilitator the agents, which could provide these atomic checks. Then it interacts with the corresponding probe agents for getting the wished atomic check.

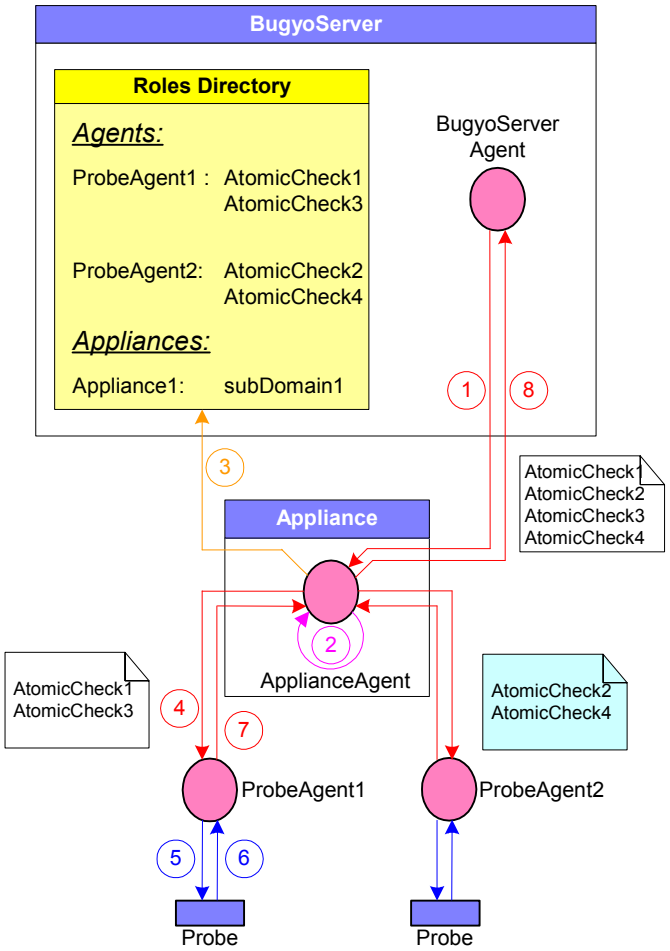

Figure 15 – Agents management

3.1.8.1 Use case

This use case presents the agent's organization; the first four steps describes what happens when each agent are started and the others steps describes the agent's behaviours when the user request to check the assurance level.

Steps:

[1] Launch BugyoServer agent and MAS Control, which will help us to manage the multi-agent platform

[2] Launch Appliance agent and register his sub-domain into DF (Directory Facilitator)

[3] Launch every probe agents

[4] Each probe agent reads his command file to extract the ID of the measurements (atomic check) that he is able to perform. The probe agent will then register every ID into the DF.

[5] User request to get the security function assurance rate of a given metric

[6] BugyoServer agent converts the demanded metric to server-appliance format (XML File) or simply uses an existing metric in server-appliance format

[7] Metric sent to Appliance agent of the desired sub-domain

[8] Appliance agent consults DF in order to know which probe agent is able to perform a given atomic check.

[9] Appliance agent dispatches the metric and sends to the different probe agent the part of the metric, which concerns only each agent.

[10] Two possible scenarios:

Scenario A: The probe agent already has the results of the measurements. This is the case when the probe is continuously sending data to his specific agent.

Scenario B: The probe agent will ask the probe to perform the measurements and send them back.

[11] Each probe agent sends the results back to appliance agent.

[12] Appliance agent gathers the parts of the metric received from each probe agent and send it back to the Server agent

[13] Aggregation of the result is done and present to the final user

3.2 Agent Design Pattern

This agent design pattern was been defined for this project.

BUGYO specifications explain that BUGYO must provide to the probe editors the way to implement easily their own BUGYO related agent. An Agent Design Pattern is the only way to standardize BUGYO agents, even if they are specific to a type of probe.

An agent design pattern offers:

1. Friendly APIs (Application Programming Interfaces) to easily create BUGYO formatted data
2. A standardization of the specific APIs for all agents

The final aim of an Agent Design Pattern is to provide to the probe editors a BUGYO Agent Development Kit (BADK) that would be a kit to easily create BUGYO agents.

The BUGYO Agent Development Kit would contain:

1. A Java library
2. APIs Documentation
3. Examples, tutorial…

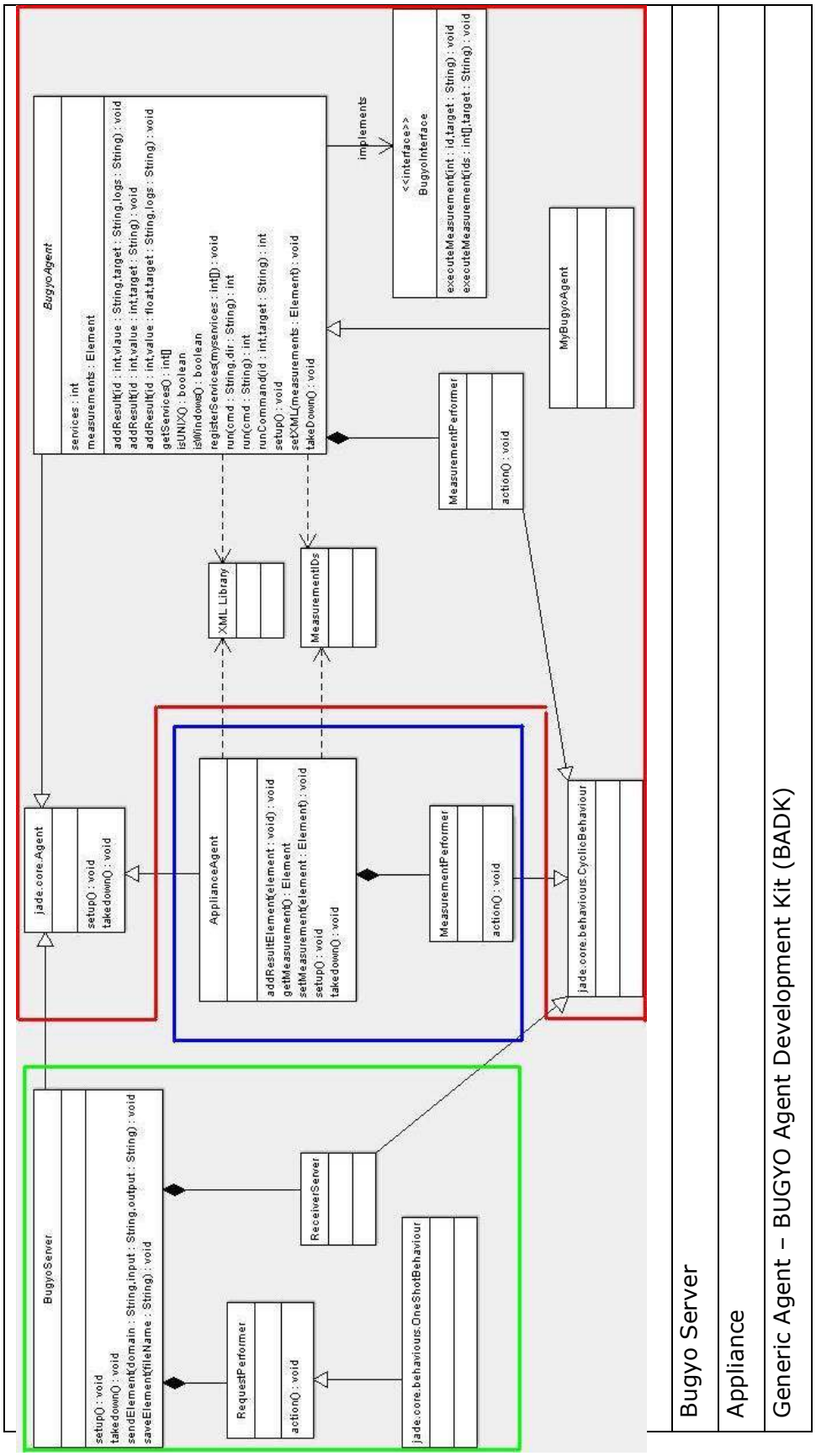
Generic Agent – BUGYO Agent Development Kit (BADK)
Appliance
Bugyo Server
BugyoAgent
services : int
measurements : Element
addResultId : int,value : String,target : String,logs : String) : void
addResultId : int,value : int,target : String) : void
addResultId : int,value : float,target : String,logs : String) : void
getServices() : int[]
isUNIX() : boolean
isWindows() : boolean
registerServices(myservices : int[]) : void
run(cmd : String,dir : String) : int
run(cmd : String) : int
runCommand(id : int,target : String) : int
setup() : void
setXML(measurements : Element) : void
takeDown() : void
implements
<<interface>>
BugyoInterface
executeMeasurement(int : id,target : String) : void
executeMeasurement(ids : int[],target : String) : void
MyBugyoAgent
MeasurementPerformer
actionO : void
XML Library
MeasurementIDs
jade.core.Agent
setupO : void
takedownO : void
ApplianceAgent
addResultElement(element : void) : void
getMeasurementO : Element
setMeasurement(element : Element) : void
setupO : void
takedownO : void
MeasurementPerformer
actionO : void
jade.core.behaviours.CyclicBehaviour
BugyoServer
setupO : void
takedownO : void
sendElement(domain : String,input : String,output : String) : void
saveElement(fileName : String) : void
ReceiverServer
RequestPerformer
actionO : void
jade.core.behaviours.OneShotBehaviour

Figure 16 - Class diagram

3.3 BUGYO Agent

3.3.1 DESCRIPTION

This agent implements all methods needed to communicate with other agents, in our case the Appliance agent. The agent also implements a set of methods, which allows the agent to register his services, treat a part of a metric (represented by an XML element), this means read and complete a XML element and also run specifics probe's commands.

This is a generic agent, all probe agents (also called specific agents) extends from this Bugyo Agent.

3.3.2 METHODS IMPLEMENTED IN THE BUGYOAGENT CLASS

- setup():void
- takedown():void
- addResult(int id, String value, String target, String logs):void

  For this function we use the overloading method of java to declare three different functions with the same name, but a different signature. The difference between each function is the type of parameter value (String, int, float).
- registerServices(int [] myServices):void
- getServices():int[]
- isUnix():boolean
- isWindows():Boolean
- run(String cmd):int
- run(String cmd, String dir):int
- runCommand(int id, String target):void
- setXML(Element measurements):void

A detailed description of these methods can be found in appendix 9.1.1.

3.3.3 BEHAVIOURS IMPLEMENTED IN THE BUGYOAGENT CLASS

Name: *PerformMeasures*

Description: *This behaviour reads sub elements received from Appliance, request the specific probe to execute the measurements, reads and treats the result of the measurement in order to complete the sub element and send it back to the Appliance.*

Steps:

[1] Receive sub element from Appliance

[2] Perform measures and complete sub element

[3] Send back the sub element to Appliance

3.4 BUGYO Interface

3.4.1 DESCRIPTION

This interface contains two methods that are specific to a probe agent. This means, that one specific agent has to implement at least one method defined in the Bugyo Interface. The implementation of these methods describe the behaviour how to execute a measurement, it could call typically methods as *runCommand()* and *addResult()* defined and implemented in BugyoAgent.

3.4.2 METHODS IMPLEMENTED IN THE BUGYOINTERFACE CLASS

- executeMeasurement(int id, String target): void
- executeMeasurement(int[] id, String target): void

A detailed description of these two methods can be found in appendix 9.1.2.

3.5 Appliance Agent

3.5.1 DESCRIPTION

This agent implements all methods needed to communicate with other agents, in our case the BugyoServer agent and probe agents. The agent also implements a set of methods, which allows the agent to register the domain name where the appliance is installed, receiving and sending one or more metrics (represented by an XML element) to BugyoServer agent, dispatching the metric and sends to the different probe agent the part of the metric, which concerns only each agent and finally gathers the parts of the metric received from each probe agent and send it back to BugyoServer agent.

3.5.2 METHODS IMPLEMENTED IN THE APPLIANCEAGENT CLASS

- dispatchXML(Element element):void
- addResultElement(Element element):void

A detailed description of these two methods can be found in appendix 9.1.3.

3.5.3 BEHAVIOURS IMPLEMENTED IN THE APPLIANCEAGENT CLASS

Name: *measurementPerformer*

Description: *This behaviour reads elements received from Bugyo Server, creates sub elements, which will be sent to each Bugyo Agent, and finally completes the input element using the sub elements to send it back to the Bugyo Server.*

Steps:

[1] Receive element from Bugyo Server
[2] Dispatch element into sub elements to send to the Bugyo Agent
[3] Receive the sub elements from each Bugyo Agent
[4] Complete the element using the sub elements
[5] Send back the completed element to Bugyo Server

3.6 Server Agent

3.6.1 DESCRIPTION

When requested from user (cockpit), this agent sends an element (representing the measurements to perform) to Appliance agent, waits for receiving the measurement's results and presents this result to the BUGYO Cockpit.

3.6.2 METHODS CONTAINED IN THE SERVERAGENT CLASS

- setMeasuremts(String filename):void
- saveElement(String filename):void
- sendElement(String domainToCheck, String inputFilename, String outputFilename):void

A detailed description of these methods can be found in appendix 9.1.4.

3.6.3 BEHAVIOURS IMPLEMENTED IN THE SERVERAGENT CLASS

Name: *RequestPerformer*

Description: *This behaviour sends elements to Appliance agent; it is called by every request from BUGYO cockpit.*

Name: *ReceiverServer*

Description: *This behaviour reads elements received from Appliance agent; it is always listening for new incoming elements.*

4 TECHNOLOGICAL SOLUTIONS IDENTIFICATION

4.1 Multi Agent System (MAS)

4.1.1 INTRODUCTION

This chapter contains a short overview of a Multi-Agent System (MAS), the results of a survey about three MAS (Jade, Aglet and Tryllian), a comparison matrix and finally a conclusion giving the chosen platform for BUGYO project.

4.1.2 MAS OVERVIEW

A **Multi-Agent System** (MAS) is a system composed of several agents, capable of mutual interaction. The interaction can be in the form of message passing or producing changes in their common environment. The agents can be autonomous entities, such as software agents or robots. MAS can include human agents as well. Human organizations and society in general can be considered an example of a multi-agent system.

Many definition of an agent exist. An **agent** is considered as a physical or virtual entity. The common characteristics of an agent entity are listed below:

- It is capable of acting in an environment.
- It can communicate directly with other agents.
- It is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize).
- It possesses resources of its own.
- It is capable of perceiving its environment (but to a limited extent).
- It has only a partial representation of its environment (and perhaps none at all).
- It possesses skills and can offer services.
- It may be able to reproduce itself.
- Its behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation and the communications it receives.

An agent, like an object, encapsulates a state and a behaviour, but:

- An agent has control on its behaviour; and has only control on its state.
- An agent exert this control in various manners (reactive, directed by goals, social)
- MAS have several control flows while a system with objects has a priori only one control flow.

The agents will have global behaviour into the MAS, such as:

- Cooperation: agents share the same goal
- Collaboration: agents share intermittently the same goal,
- Competition: incompatible goals between agents

A foundation, called Foundation for Intelligent Physical Agents (FIPA) [5], promotes the success of emerging agent-based applications, services and equipment. Making available in a timely manner, internationally agreed specifications that maximize interoperability across agent-based applications, services and equipment pursues this goal. This is realized through the open international collaboration of member organizations, which are companies and universities active in the agent field. FIPA's specifications are publicly available. They are not a technology for a specific application, but generic technologies for different application areas, and not just independent technologies but a set of basic technologies that can be integrated by developers to make complex systems with a high degree of interoperability.

Many agents' communications languages exist. One of them is KQML (Knowledge Communication Meta Language). It was one of the earliest attempts of construct an agent communication language based on speech act theory. It has had a major influence on later developments.

FIPA proposed ACL (Agent Communication Language), based on KQML. It represents the world standard for agent communication. ACL is closely related to KQML. In fact, ACL represents a smaller version of KQML, which is more precisely defined. XML has become quite popular. There is a standard version of ACL written in XML and sponsored by FIPA.

4.1.3 DIFFERENT MAS PLATFORMS

In this section, several well-known MAS platforms are described and compared regarding some criteria, such as performance, maturity, architecture and communication technologies, etc.

4.1.3.1 JADE

Platform Description

JADE (Java Agent DEvelopment Framework) [6] is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middleware, which is FIPA compliant.

The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of JAVA (the run time environment or the JDK).

Libraries, called LEAP (Lightweight Extensible Agent Platform) [6] combined with the JADE platform, provide a run-time environment with reduced footprint and suitable for mobile lightweight Java environments. LEAP is the name of the European IST project that has developed the LEAP libraries.

Platform Architecture

JADE ensures standard compliance through a comprehensive set of system services and agents in compliance with the FIPA specifications: naming service and yellow-page service, message transport and parsing service, and a library of FIPA interaction protocols ready to be used.

The AMS (Agent Management System) provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS).

The DF (Directory Facilitator) provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

The ACC (Agent Communication Channel) is a high-level interface, through which messages are sent from one agency to another, using an MTP (Message Transport Protocol). The Message Transport Protocol (MTP) carries out the physical transfer of messages between two ACCs.

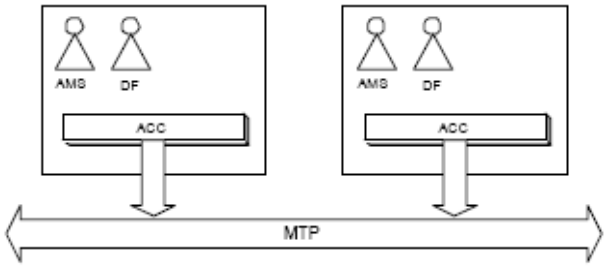

Figure 17 - JADE Agent Communication Channel

All agent communication is performed through message passing, where FIPA ACL is the language to represent messages. The following protocols are used:

- Intra platforms, Intra containers: Java Events
- Intra platforms, Inter containers: RMI
- Inter Platforms: IIOP Corba, HTTP

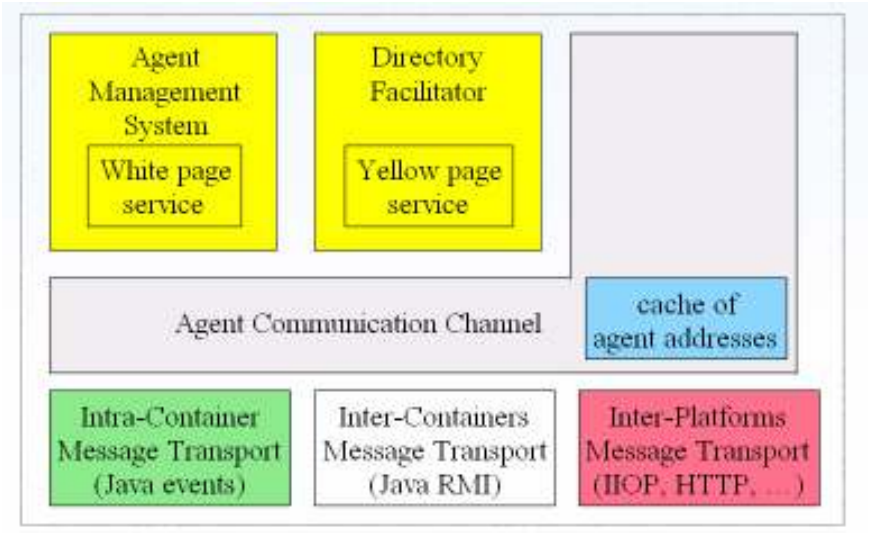

Figure 18 - JADE Platform Architecture

Basically, agents are implemented as one thread per agent, but agents often need to execute parallel tasks. Further to the multi-thread solution, offered directly by the JAVA language, JADE supports also scheduling of cooperative behaviours, where JADE schedules these tasks in a light and effective way. The run-time includes also some ready to use behaviours for the most common tasks in agent programming, such as FIPA interaction protocols, waking under a certain condition, and structuring complex tasks as aggregations of simpler ones.

Agents Deployment

The agent platform can be distributed on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is basically a container of agents that provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host.

Besides the ability of accepting registrations from other containers, a main container differs from normal containers as it holds two special agents (automatically started when the main container is launched).

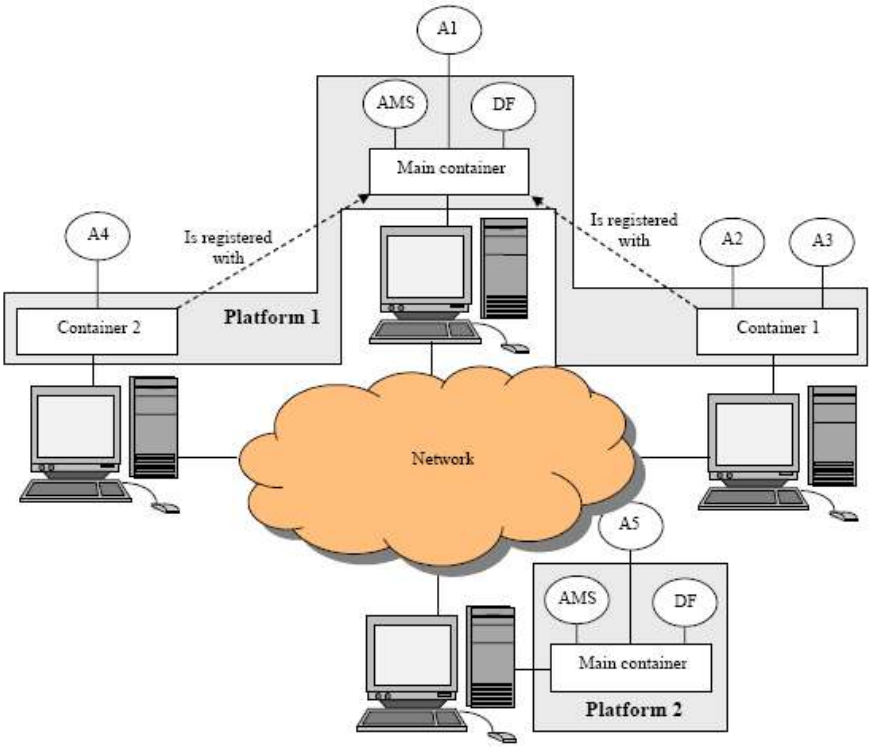

Figure 19 – JADE Agent Deployment

Agents Communication

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching based.

The full FIPA communication model has been implemented and its components have been clearly distinct and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and, finally, transport protocols. The transport mechanism, in particular, is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, HTTP, and IIOP are currently used, but more protocols can be easily added via the MTP and IMTP JADE interfaces.

Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application-dependent behaviour of each state of the protocol. SL and agent management ontology have been implemented already, as well as the support for user-defined content languages and ontologies that can be implemented, registered with agents, and automatically used by the framework.

Agents Mobility

Intra-Platform

The two public methods doMove() and doClone() of the Agent class allow a JADE agent to migrate elsewhere or to spawn a remote copy of itself under a different name. Method doMove() takes a jade.core.Location as its single parameter, which represents the intended destination for the migrating agent.

Moving an agent involves sending its code and state through a network channel, so user defined mobile agents must manage the serialization and unserialization process.

Inter-Platform

The Inter-Platform Mobility Service has been designed and implemented to provide platform-to-platform mobility for JADE agents, a feature which is not available with the JADE built-in Agent Mobility Service.

The basic idea behind this service is moving agents from platforms to other platforms carrying them inside FIPA ACL Messages. These messages are sent between AMS agents of the involved platforms and follow the rules imposed by some of the FIPA Interaction Protocols.

Platform Security

In order to achieve secure communication, both confidential and mutual authenticated, between JADE containers of the same agent platform, it is necessary to add some components, as shown in the figure below.

Figure 20 - JADE Platform Security

Main, cont1: JADE containers of the same agent platform.

keystore : It is a file containing: an asymmetrical key-pair and a self-signed certificate. Access to this file's content is protected by a password (aka passphrase). This file must never be disclosed to others. To certain extent, it can be seen as equivalent to a passport, a credit card or a smart-card.

TrustStore : It is a file containing a list of trusted parties. It contains a list of self-signed certificates of all containers who are known and trusted. If a container tries to connect and his certificate is not included here, connection is rejected.

4.1.3.2 Tryllian Agent Development Kit

Platform Description

The Tryllian Agent Development Kit (ADK) [8] is a mobile component-based development platform that allows Java Developers to build, deploy and manage secure, large-scale distributed solutions that operate regardless of location, environment or protocol.

The ADK features remote management across platforms, customization and flexibility. It excels wherever code needs to be updated remotely, needs to perform monitoring or assisting functionality in intermittently connected situations.

One of the innovations of the ADK, is task-oriented programming. When building an ADK application, tasks have to be defined: high-level active building blocks that can be combined together in an intuitive and manageable way to create complex

logic. Typical tasks and applications of mobile agents include remote systems monitoring and management, distributed application integration (with agents wrapping the various applications or data sources), workflow automation, dynamic deployment of (native) code and components, and more.

Platform Architecture

Tryllian is a distributed peer-to-peer (P2P) component architecture with XML message-based communication supports JMS, JXTA and web services concurrently. It uses a P2P library called JXTA [9]. Project JXTA is a P2P-based networking technology from Sun Microsystems.

It is also based JNDI directory services and a lightweight runtime environment based on Java 2 Standard Edition. Agents can find each other using a Naming Service that, in turn, uses the JNDI (javax.naming) interface.

Agents Deployment

Based on a peer-to-peer architecture, Tryllian allows automatic configuration for dynamic discovery, adding and removal of network nodes and services.

Agents Communication

Agents use String-based messaging to communicate with each other; this is done both for security reasons and compatibility reasons.

By default the ADK uses a HTTP based protocol to transfer messages between habitats and agents containers. Thanks to a plugin messenger architecture and messaging gateway tasks, alternative protocols such as SOAP and JMS are or can become supported.

Agents Mobility

A typical application has a number of locations with specific roles (e.g. server machine, database machine etc.), but agents can also be dynamically directed to specific locations. Scenarios where agents form a peer-to-peer network or perform network discovery are also possible.

Agents offer themselves to the Transporter System Agent. The Transporter encodes the agent's code and state into a set of messages and uses the transport layer to communicate with the Transporter at the destination.

Agent code is only transmitted when necessary (not if already present); code identity is based on several aspects, among which include a secure hash code over the Jar files. This ensures an optimal transport time, without requiring the code to be present on the other side. Multiple versions of the same Java class can co-exist in a habitat, eliminating a number of version control issues.

Platform Security

Agents are authenticated and authorized at a class level, following the Java 2 Security Model. The identity of an agent is established based on the identity of the entity signing the agent's classes. Signing agent's classes is performed using a public-private key method. The algorithm used for this is DSA with SHA1, with key length of 512, 768 or 1024 bits.

The habitats (agent runtime environments) keep a list of known issuers (certification authorities) and signers to solve trust issues. We use standard X509 certificates; they can be issued by Tryllian or other parties.

The following additions were made to the Java Security Model:

- Permissions are grouped in sets, called "roles". Roles can be assigned to identities, based on the owner or issuer of the certificate. The habitat administrator grants roles. This allows you to group related entities into one permission set.
- Agents need explicit permissions to enter a habitat. Once in the habitat, agents need explicit permission to access system resources, e.g. to create threads, access files, or use network connections.

Habitat communication is done using HTTP or HTTPS. When using HTTPS, only mutually authenticated habitats will communicate. By default, habitat authentication is done using DSA using a key length of 412, 768 or 1024 bits. Messages exchanged by the habitats are encrypted and their integrity is protected by a TLS based protocol.

4.1.3.3 Aglet [7]

Another partner elaborated this section.

4.1.3.4 Platform Comparison Matrix

Platform	JADE	Aglet	Tryllian
General			
Creation date	Open Source since	1995	Founded in 1998, Open-

	February 2000 (version 1.3)		Source on 15 November 2005
Official web site	http://jade.tilab.com/	http://www.trl.ibm.com/aglets	http://www.tryllian.com/
Latest official version	3.4 (14/03/06)	2.0.2 (19/02/02)	3.2.0
Developed by	Tilab (Telecom Italia R&D Center), It	IBM Tokyo Research Laboratory (Japan)	Tryllian (NL)
Language	Java	Java	Java
License	LGPL	IBM Public License	LGPL
Board	TILAB, Motorola, Whitestein Technologies AG, Profactor GmbH, and France Telecom R&D.	TabiCan (Japan)	Unknown
Mailing list	jade-news@sharon.cselt.it	aglets-users@lists-sourceforge.net aglets-commits@lists.sourceforge.net aglets-developer@lists.sourceforge.net	adk-users@tryllian.org adk-developers@tryllian.org
Forum	jade-develop@sharon.cselt.it		
Documentation	http://jade.tilab.com/doc/index.html	User manual: http://ovh.dl.sourceforge.net/sourceforge/aglets/manual.pdf http://aglets.sourceforge.net/doc/index.html	http://www.tryllian.org/documentation.html

		specification : http://www.trl.ibm.com/aglets/spec11.htm	
Supported platforms	Java 2, J2EE, J2ME	Java 2, J2EE, J2SE, (J2ME ?)	Java 2, J2ME
FIPA Compliance	Yes	No, MASIF compliant only	Yes
Agents Communication			
Messages' Language or Format	FIPA Agent Communication Language (ACL)	String Message	Agents use String-based messaging to communicate with each other; this is done both for security reasons and compatibility reasons.
Communication Protocol	Java events, RMI, HTTP	ATP (Agent Transfert Protocol), HTTP, FTP, RMI, CORBA	HTTP SOAP and JMS could be supported using a plugin messenger architecture and messaging gateway tasks.
Mode	Synchronous and asynchronous	Synchronous and asynchronous	?
Security			
Security Model	Based on principals, resources and permissions, which enable authentication and authorization of both agents and the owners.	Based on roles (aglet, manufacturer and owner). Authentication and authorization mechanism are available. Integrity could be also checked. Permissions could also refine the security policy.	Permission-based operation that prevents illegal entry or operations.
Encryption between Agent-Platform	SSL / TLS	SSL	SSL / TLS

Communication			
Encryption between Agent-Agent Communication	SSL /TLS	SSL	SSL / TLS
Authentication between Agent-Platform Communication	Based on JAAS (Java Authentication and Authorization Service, http://java.sun.com/products/jaas)	X509 certification	X509 certification
Authentication between Agent-Agent Communication	Based on JAAS	Based on JASS	X509 certification
Others			
Mobility	Yes	Yes	Yes All agents in the habitat are check pointed to the database, as well as the state of the JNDI registry.
Managing Concurrency	Yes	Yes	Yes
Performance	Send 100kb message from an agent to another in 90ms (see figure in Appendix and [6])	?	Send 100kb message from an agent to another in 380ms (see figure in Appendix and [6])
Scalability	By increasing the numbers of agents-pairs to 100, jade still running (see figure in Appendix and [6])	By increasing the numbers of agent-pairs to 5000, Aglet is still running. 5000 agents have migrated from a host to	Testing 100 agents on each machine Tryllian failed (see figure in Appendix and [6])

		another without troubles.	
Persistency	Allow saving and reloading agent state on relational DB (Based on Hibernate)	After a migration, agent's state is reloaded.	The ADK offers database persistence to allow a running habitat/application to continue after it has been restarted. Agents reside on a network, and in the event that the network, and thus the habitat, goes down, the ADK provides persistence to prevent the agents from disappearing
Platform Management Tools	RMA (Remote Agent Management), DF, Sniffer Agent, Dummy Agent Introspector Agent,	Tahiti (GUI) and a command line tool.	-

4.1.4 CONCLUSION

A Multi-Agent platform offers many advantages, which could be profitable for BUGYO framework: Autonomy, Intelligence, Mobility, Robustness, Scalability, Flexibility, API already presents and communication language and architecture already defined. Using agents in distributed computing offers the possibility to support heterogeneous systems, adaptability and loose coupling. After the necessary measurements have been performed on a NE, an agent could move to another NE to perform the same type of measurements. Although important, mobility is not a crucial criterion for BUGYO.

This document investigated Multi-Agent System (MAS) and Platform. Several platforms were described.

Aglet and JADE architectures have a lot of technical similarities, and both offer the same functionalities concerning agents (for instance migration), security (authentication, confidentiality, etc.). JADE's architecture is a centralized architecture. For Aglet, there is only on type of container, it implies the architecture is not centralized in a single container.

Tryllian is peer-to-peer based. Particularly for security reasons in the targeted networks, peer-to-peer architecture is not appropriated. Consequently Tryllian could not be used in BUGYO.

In term of maturity, even if Aglet (creation in 1995) is older than JADE (creation in 2000), both platforms offers a large amount of documentation, mailing list and forums. Tryllian was at the beginning a commercial products, but now open-source.

In term of performance, different tests already leaded in this direction, show that JADE offers better results than others.

At the security level, the three platforms offer very good features to set up a safe communication between agents.

Furthermore in term of dynamism, standardization, prestige, and license JADE and Aglet have many differences. Dynamism is the ability to perform, update the project. The last version of JADE has been updated in March 2006, whereas the last version of Aglet is dated from February 2002. So Jade is more dynamical than Aglet.

Concerning standardization, Aglet is MASIF compliant whereas JADE is FIPA compliant. FIPA and MASIF are two foundations for agent standardization. MASIF is older than FIPA, but FIPA tend to be more and more the standard approved in agent architectures. So, for BUGYO it would be more interesting to be FIPA compliant than MASIF compliant.

Well-known companies, such as France Telecom, use JADE.

Finally JADE is licensed under the LGPL license, which will facilitate its integration in a product like BUGYO.

As a conclusion, JADE platform responds more to the BUGYO expectations.

4.2 Probes

4.2.1 OVAL

4.2.1.1 General Information

Open Vulnerability and Assessment Language (OVAL) [10] is an international, information security community baseline standard for how to check for the presence of vulnerabilities and configuration issues on computer systems.

OVAL standardizes the three main elements of the checking process:

- A framework to manage the collection of configuration information from systems
- A language to structure the tests that determine the presence of specific vulnerabilities, configuration issues, and/or patches on a system
- A common format for presenting the results of the tests against a specific system

For each of these elements, the OVAL Community, a collection of information security experts from industry, academia, and government dedicated to developing and promoting security standards, has developed Extensible Markup Language (XML) schemas to serve as the framework and vocabulary of OVAL.

OVAL definitions provide a detailed method for checking low-level configuration parameters on a computer to determine the presence or absence of software vulnerabilities, but OVAL doesn't include information about vulnerabilities, such as the severity of the problem, whether it is locally or remotely exploitable, remediation information, and so on.

4.2.1.2 OVAL Board

The OVAL Board was created to coordinate the efforts of the OVAL Community and to provide focus for the development of the language. The OVAL Board consists of representatives from major operating system vendors, commercial information security tool vendors, academia, government agencies, and research institutions, who are on the Board because of their expertise and leadership in the information security community.

The OVAL standard initiative was funded by US-CERT at the U.S. Department of Homeland Security,

4.2.1.3 The OVAL Schemas Overview

There are 3 categories of OVAL schema:

The OVAL System Characteristics Schema

The OVAL System Characteristics defines a standard XML format for storing tested system configuration information (Operating System parameters, Installed software, Application settings,...

The OVAL Definition Schema

THE OVAL Definition Schema defines a standard XML format for writing 3 types of threats:

- **OVAL Vulnerability Definitions:** defines the conditions that must exist on a computer for a specific vulnerability to be present
- **OVAL Patch Definitions:** defines the conditions that must exist on a computer that determine whether a particular patch is appropriate
- **OVAL Compliance Definitions:** defines the conditions on a computer that determine compliance with a specific policy or configuration statement

The OVAL Vulnerability Definitions are based on Common Vulnerabilities and Exposures (CVE), which is a worldwide dictionary of standardized names of vulnerabilities discovered. Each OVAL Definition Schema refers to a single CVE-ID. The CVE database can be found on: http://cve.mitre.org.

The OVAL Results Schema

The OVAL Results Schema defines a standard XML format for storing the results of an OVAL evaluation of a system.

The Document Type Description (DTD) of these different OVAL schemas can be found on the official site on the page:

http://oval.mitre.org/oval/download/schema/index.html

4.2.1.4 Content of the OVAL Schemas

Content of the OVAL Definition Schema

The Definition Schema contains:

- One definitions element containing one or more definition elements
- One tests element containing one or more test elements

The **Definition** elements contain:

- An **id** attribute: a unique ID assigned by the MITRE Corporation, which has the form 'OVAL' followed by a number of digits.
- A **class** attribute: indicates the specific class to which the definition belongs. Possible classes are: 'compliance', 'patch', and 'vulnerability'.
- A single **affected** element with the **family** attribute: the **family** attribute refers to the name of the affected product family (e.g. "Windows"). At this time, 15 families are referenced: Aix, Apache, Debian, FreeBSD, HP-UX, ios, Macos, OpenBSD, Oracle, OS400, Pix, RedHat, Solaris, Suse and Windows.
- The **affected** element contains a collection of **platform** elements corresponding to the vulnerable platforms. (e.g. "Microsoft Windows 2000", "Microsoft Windows XP")
- The **affected** element also contains a collection of **product** elements: names of the application for which the Definition file has been created. (e.g. 'Microsoft Excel 2000').
- A single **dates** element: schema containing date of submission of the definition, history of the status…
- A **description** element: containing a brief description of the vulnerability. Generally corresponding to the description of the CVE reference.
- Zero or more **reference** elements: CVE ID for which the Definition file has been written. Most of the time, a Definition file refers to one and only one CVE advisory.
- One **status** element: indicates the status of the Definition. Could take only five values: ACCEPTED, DEPRECATED, DRAFT, INCOMPLETE, INITIAL SUBMISSION or INTERIM.
- One **version** element: containing the version of the Definition file. Versions are integers, starting at 0 and incrementing every time a Definition reaches ACCEPTED status.
- Zero or One **criteria** element: **criteria** element is the high level of all tests and represents the meat of the Definition file. A **criteria** element contains one or more **criterion** elements.

- The **criterion** elements identify a single test (contained in the **test** element) with a unique ID called ***test_ref***. The criterion element is also containing a brief description of the test. The real implementation of the test will be set in the **test** element.

The Test elements

In fact the **test** element is an abstract element, which has to be extended by the tests found in the specific test families. There are 9 families of platform specific tests:

- Cisco – IOS specific tests
- Debian Linux specific tests
- HP – UX specific tests
- Macintosh specific tests
- Red Hat Linux specific tests
- Solaris specific tests
- Unix specific tests
- Windows specific tests
- Independent specific tests

Each specific test has the suffix '**_test**'. For example, the **registry_test** test a specific registry key, **port_test** test information about listening ports…

Although these tests are platform dependent, it exist great test families which can be found in the majority of supported platforms:

Test	**Description**
File test	Checks the file's metadata.
Interface test	Enumerates various attributes about the interfaces on a system.
Permission test	Gives information about the hardware the machine is running on.
Uname test	Gives information about the hardware the machine is running on.
Process test	Process test: checks the process information for a given process.

File content test	Access information stored in configuration files.
XML content file test	Access information stored in XML files.
Group test	Tests users that belong to specific groups
Password policy test	Test specific policy associated with passwords.
Port test	information about listening ports.
Registry test	test a specific registry key.

The complete list of supported tests grouped by platform can be found on the page: http://oval.mitre.org/oval/about/documents.html

Content of the OVAL System Characteristics Schema

The System Characteristic Schema contains:

- A single **system_info** element
- Zero or one **system_data** element

A **system**_info element contains:

- The operating system of the tested machine (**os_name** element)
- The version of the operating system (**os_version** element)
- The architecture of the tested machine (**architecture** element)
- The primary host name of the tested machine (**primary_hot_name** element)
- An **interfaces** element, which is an **interface** elements. Each of the **interface** elements contains the name of the interface (**interface_name**), its IP address (**IP_address**) and its MAC address (**mac_address**).

A **system_data** element contains results of all tests present in the OVAL Definition file.

Content of the OVAL Result Schema

The Result Schema contains:

- A single system_info element containing information extracted from the System Characteristics file: OS, version, architecture, interfaces…)

- Zero or one definitions element containing a collection of definition elements.
- Zero or one tests element containing a collection of test elements. The tests element acts as a container for the detailed results of each test required by the specified OVAL definition.

For each definition element contained in the Definition Schema, a definition element is created in the Result Schema.

In the Result Schema, the definition element contains the same elements than in the Definition Schema adding:

- a **result** attribute to the criteria element
- a **result** attribute to the criterion element

In both cases, the result attribute indicates if the test passed (true, false or error). The result attribute in the criteria element indicates if the system is vulnerable or not, whereas the result attribute in the criterion element indicates the result for each test.

4.2.1.5 Interaction between the different OVAL schemas

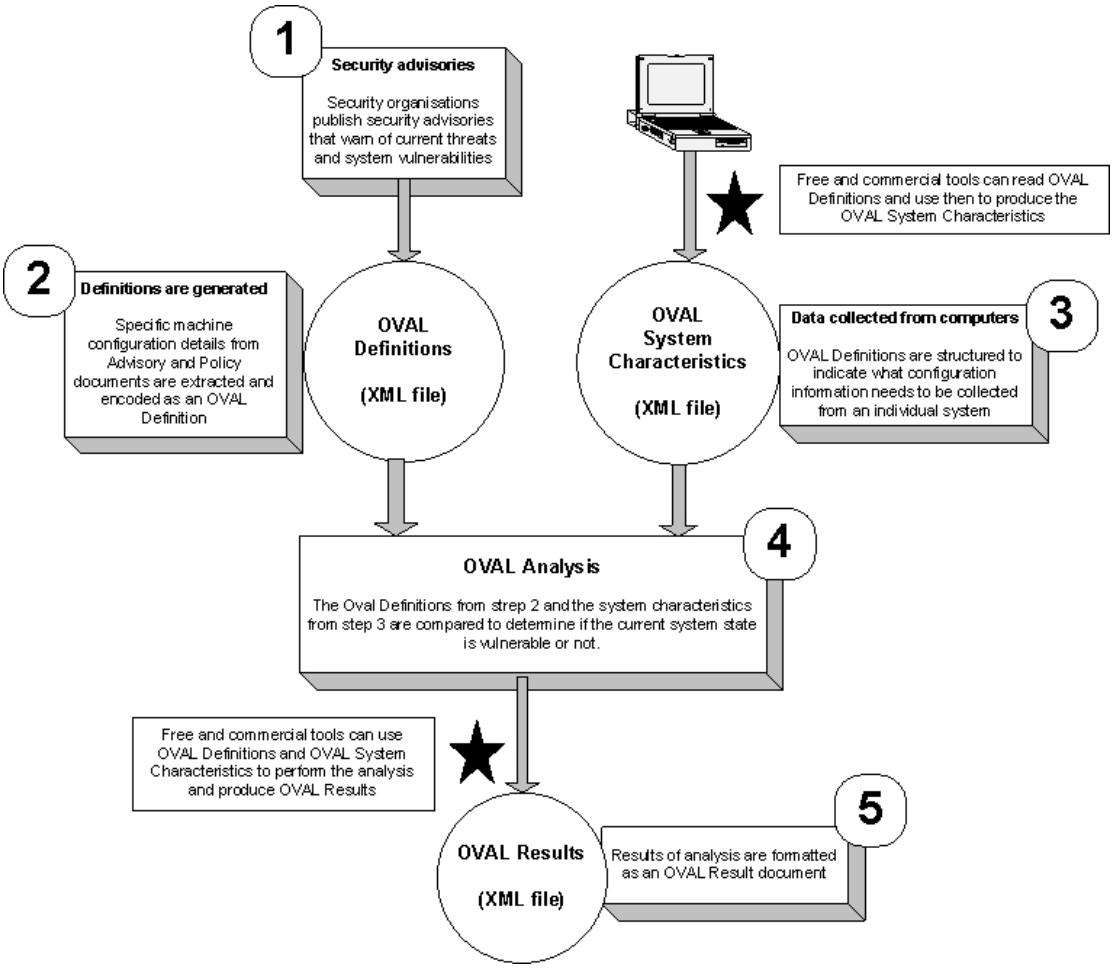
1
Security advisories
Security organisations publish security advisories that warn of current threats and system vulnerabilities
Free and commercial tools can read OVAL Definitions and use then to produce the OVAL System Characteristics
2
Definitions are generated
Specific machine configuration details from Advisory and Policy documents are extracted and encoded as an OVAL Definition
OVAL Definitions
(XML file)
OVAL System Characteristics
(XML file)
Data collected from computers
3
OVAL Definitions are structured to indicate what configuration information needs to be collected from an individual system
4
OVAL Analysis
The Oval Definitions from strep 2 and the system characteristics from step 3 are compared to determine if the current system state is vulnerable or not.
Free and commercial tools can use OVAL Definitions and OVAL System Characteristics to perform the analysis and produce OVAL Results
OVAL Results
(XML file)
5
Results of analysis are formatted as an OVAL Result document

Figure 21 - How to use OVAL

Step 1: Security threats are discovered and referenced to CVE, which is a worldwide dictionary containing all advisories. Each advisory get a unique ID called CVE-ID.

Step 2: For each CVE-ID, an OVAL Definition file containing the CVE-ID, a brief description of the vulnerability, the target platform and the configuration (installed software, applications settings…) that determine the vulnerability is created.

Step 3: For each OVAL Definitions file, characteristics of the tested system are searched. The OVAL System Characteristics is created.

Step 4: Characteristics of the tested system and characteristics needed for the vulnerability are compared. If they are the same, it means the system is vulnerable.

Step 5: Finally, a report containing all vulnerabilities discovered on the tested system is created. The OVAL Results file is created.

4.2.1.6 OVAL Definition Interpreter (ovaldi)

MITRE's OVAL Definition Interpreter is a freely available reference implementation created to show how information can be collected from a computer to evaluate, and carry out, the OVAL definitions for that platform.

MITRE developed the Reference Definition Interpreter to demonstrate the usability of OVAL definitions, and for definition writers to use to ensure correct syntax and adherence to the OVAL Schemas during the development of draft definitions. It is not a fully functional scanning tool and has a simplistic user interface, but running the Definition Interpreter will provide you with a list of OVAL-IDs and their references (e.g., CVE names) determined by OVAL to be present on the system.

More information about how to define your own definition file and to use OVAL Definition Interpreter can be finding in appendix 9.2 and 9.3.

4.2.2 NAGIOS

Nagios® [11] is a system and network monitoring application. It watches hosts and services that you specify, alerting you when things go bad and when they get better.

Nagios was originally designed to run under Linux, although it should work under most other unices as well.

Some of the many features of Nagios® include:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Monitoring of host resources (processor load, disk usage, etc.)
- Simple plugin design that allows users to easily develop their own service checks
- Parallelized service checks
- Ability to define network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable

- Contact notifications when service or host problems occur and get resolved (via email, pager, or user-defined method)
- Ability to define event handlers to be run during service or host events for proactive problem resolution
- Automatic log file rotation
- Support for implementing redundant monitoring hosts
- Optional web interface for viewing current network status, notification and problem history, log file, etc.

More information about how to install and use Nagios can finding in appendix 9.4

5 IMPLEMENTATION

For the implementation, I used Eclipse 3.1 as IDE under Windows 2000 and the version 1.5.06 of the Java environment. As decided before, the multi-agent platform used is Jade 3.4.

5.1 BADK

For the implementation of the BUGYO Agent Development Kit (Generic Agent), the methods were shaded with another partner. They develop some methods in they side and I develop the others methods. For sharing these methods we used the versioning server SubVersion.

5.2 ApplianceAgent and ServerAgent

After defining the methods and the behaviours of these two agents, I have implemented them and then shared it with OPPIDA so that they also can test and develop their own specifics agents.

5.3 Specifics Agents

5.3.1 TEMPLATE – PROBE INTEGRATION SPECIFICATION

This is a draft of a template that we propose for the scenario specification in order to integrate new probes to the BUGYO agent-based platform.

During a probe's life cycle, some parameters could change. For example, some probes need authentication per login / password, or dictionary file. These kinds of information must not be directly integrated in the java code, but must be given as parameter to the agent.

Scenario

Name				
Description				
Metric Information				
Name				
Description				
Atomic check 1	ID			
	Name			
	Description			
	Automatic / Probe			
	Name			
	Description			
	OS			
	Location			
	Command prompt			
	Parameters	Parameter 1	ID	
			Value	
		Parameter 2	ID	
			Value	
		Parameter 3	ID	
			Value	
	Output	Location		
		Format description		
		Target value		
	Manual Input			
Atomic check 2	ID			
	Name			
	Description			
	Automatic / Probe			
	Name			
	Description			

OS			
Location			
Command prompt			
Parameters	Parameter 1	ID	
		Value	
	Parameter 2	ID	
		Value	
	Parameter 3	ID	
		Value	
Output	Location		
	Format description		
	Target value		
Manual Input			

Figure 22 - Template - Probe Integration Specifications

5.3.2 OVAL AGENT

This agent is able to launch OVAL Definition Interpreter to perform the requested measurements. Once the measurements finished, this agent reads the result file generated by OVAL Definition Interpreter, extracts the needed data and transforms it to agent-appliance format in order to send it to the Appliance agent.

In OVAL result file, we can extract three kinds of result:

- If criteria_result equal to 1, then the tested vulnerability is present on the system.
- If criteria_result equal to 0, then the tested vulnerability isn't present on the system.
- If criteria_result equal to -1, then the tested vulnerability wasn't unable to verify.

The next figure is representing the xml command file for the OVAL Definition Interpreter agent.

```
<measurements>
  <measurement id="0">
    <command dir="C:\OVAL\ovaldi\">ovaldi.exe -o @bugyo_definitionfile@ -r
                    @bugyo_resultfile@ -m</command>
    <parameter id="definitionfile" value="definition0.xml"/>
    <parameter id="resultfile" value="result0.xml"/>
  </measurement>
</measurements>
```

Figure 23 – Command file for OVAL Definition Interpreter probe

5.3.3 NAGIOS AGENT

This agent has to be developed when the analysis of Nagios fully completed.

5.3.4 SSL AGENT

This agent is specific to test a SSL connection. He will call several shell scripts (here considered as probe) in order to check a SSL connection. These scripts generate log files, which will be read from the specific agent to extract the result of the executed checks.

The log files contain the result (1 for OK and 0 for not OK) and description of the check.

NB: These scripts have been written and provided by Telindus.

The next figure is representing the xml command file for the SSL agent.

```
<measurements>
   <measurement id="0">
      <command dir="./BUGYO/SSL/"> AT0.script</command>
      <parameter id="logfile" value="ATO.log"/>
   </measurement>
   <measurement id="1">
      <command dir="./BUGYO/SSL/"> AT1.script</command>
      <parameter id="logfile" value="AT1.log"/>
   </measurement>
   <measurement id="2">
      <command dir="./BUGYO/SSL/"> AT2.script</command>
      <parameter id="logfile" value="AT2.log"/>
   </measurement>
 ...
   <measurement id="n">
      <command dir="./BUGYO/SSL/"> ATn.script</command>
      <parameter id="logfile" value="ATn.log"/>
   </measurement>
</measurements>
```

Figure 24- Command file for SSL agent

5.4 Graphic User Interface

For representing the measurements results, a partner (EADS) has developed the fallowing GUI, which we integrated in your agent-based application. This GUI is based on the Java Swing technology and was delivered as a panel, which can be easily integrated in an existing Swing application.

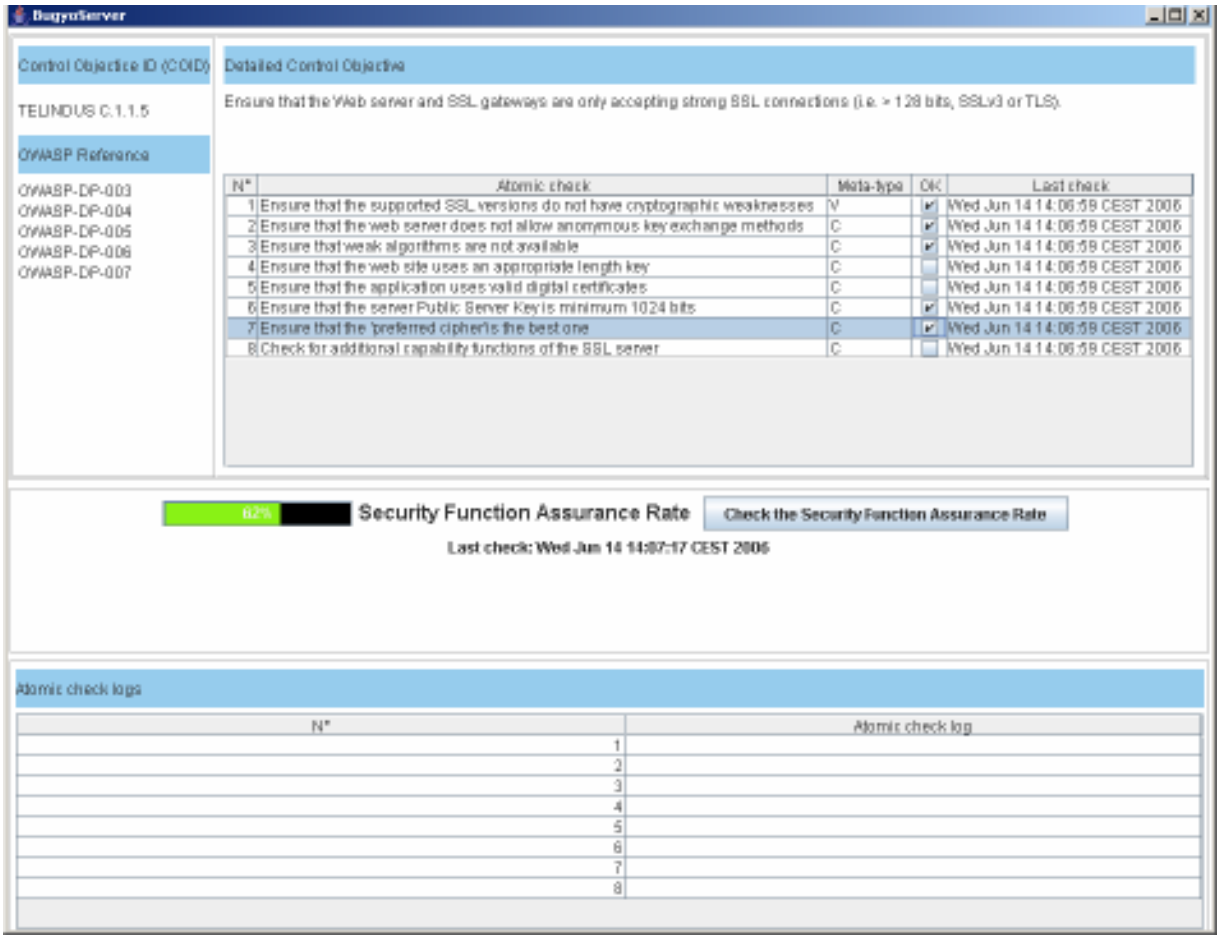
BugyuServer
Control Objective ID (COID)
TELINDUS C.1.1.5
OWASP Reference
OWASP-DP-003
OWASP-DP-004
OWASP-DP-005
OWASP-DP-006
OWASP-DP-007
Detailed Control Objective
Ensure that the Web server and SSL gateways are only accepting strong SSL connections (i.e. > 128 bits, SSLv3 or TLS).
N°
Atomic check
Meta-type
OK
Last check
1 Ensure that the supported SSL versions do not have cryptographic weaknesses V Wed Jun 14 14:06:59 CEST 2006
2 Ensure that the web server does not allow anonymous key exchange methods C Wed Jun 14 14:06:59 CEST 2006
3 Ensure that weak algorithms are not available C Wed Jun 14 14:06:59 CEST 2006
4 Ensure that the web site uses an appropriate length key C Wed Jun 14 14:06:59 CEST 2006
5 Ensure that the application uses valid digital certificates C Wed Jun 14 14:06:59 CEST 2006
6 Ensure that the server Public Server Key is minimum 1024 bits C Wed Jun 14 14:06:59 CEST 2006
7 Ensure that the 'preferred cipher' is the best one C Wed Jun 14 14:06:59 CEST 2006
8 Check for additional capability functions of the SSL server C Wed Jun 14 14:06:59 CEST 2006
62%
Security Function Assurance Rate
Check the Security Function Assurance Rate
Last check: Wed Jun 14 14:07:17 CEST 2006
Atomic check logs
N°
Atomic check log
1
2
3
4
5
6
7
8

Figure 25 - Graphic User Interface

6 TESTING

6.1 Testing Architecture

This architecture is architecture is composed of two NE using a Windows operating system and one NE using a Linux operating system. All these NE belongs to the same domain.

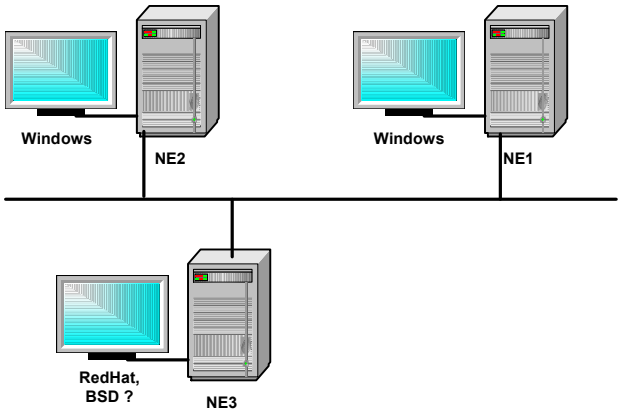

Figure 26 – Testing Architecture

6.2 BUGYO Test Platform

To test your prototype we simply deployed it in the testing architecture described above. The test platform contains some specifics agents and the corresponding probes, which agents can control. In this platform we an also see a MAS Runtime Environment per NE, which contains one or more agents, and a MAS Control Platform in the same NE as the BugyoServer agent. MAS Control Platform aims to control the multi-agent platform.

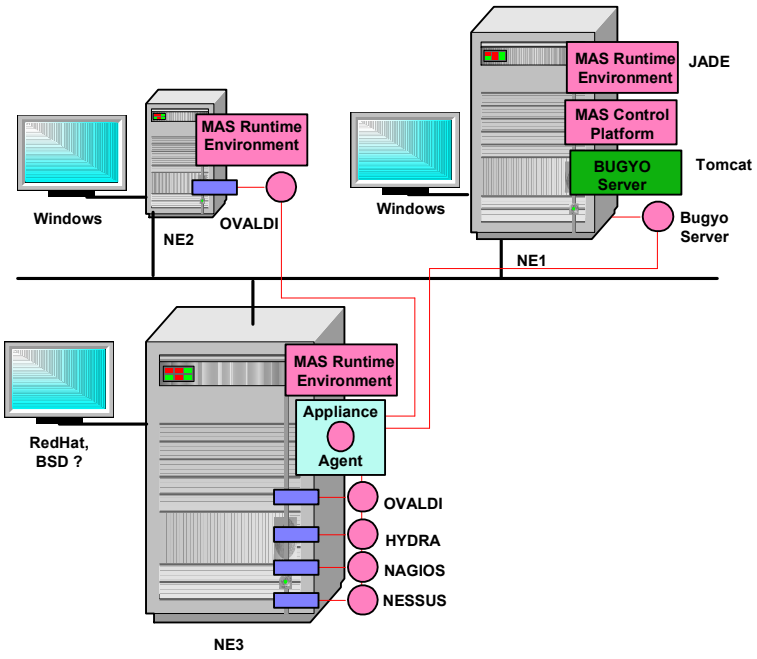

Figure 27 - Prototype Architecture

6.3 Web server with SSL

For the mid-term review in end June, we decide to demonstrate the fallowing scenario:

This demo aims to test a SSL connection and to demonstrate how the security assurance changes when changing some configuration of SSL server.

Atomic checks:

- Ensure that the supported SSL versions do not have cryptographic weaknesses
- Ensure that the web server does not allow anonymous key exchange methods
- Ensure that weak algorithms are not available
- Ensure that the web site uses an appropriate length key
- Ensure that the application uses valid digital certificates
- Ensure that the server Public Server Key is minimum 1024 bits
- Ensure that the 'preferred cipher'is the best one
- Check for additional capability functions of the SSL server

For this scenario we use the specific agent: SSL agent described in the sub chapter 6.3.4.

7 CONCLUSION

The results obtained and reported within the actual report meet the main objective of this internship that was to develop a monitoring security tool.

Different phases of the project have been established in order to be in line with the requirements specified at the beginning of the internship.

We consider that we reached the fallowing achievements:

1. Study of the scientific and technological aspect related to this project such as multi-agent system, network platform, security assurance and technological security solutions.
2. Analysis of the requirements provided at the beginning of the project and depth study in order to elaborate the design solution of the architecture to elaborated software.
3. Development of the solution based on the analysis and design. This permitted to provide a software prototype based on agent technologies.

On important aspect of the actual project is that my work was fully integrated within BUGYO project and therefore I had to respect and consider some real conditions of an industrial project (cooperation with other partners), which is a real challenge for me.

We actually have been facing some issues related to the software integration of external software modules provided with the BUGYO project partners.

The developed platform is a Java based functional solutions and is composed of:

- Client Module
- BUGYO server and its underlined sub-agents functions
- Probe middleware integration
- Implementation of a scenario developments for systems conformity checking (i.e. OVAL)

8 REFERENCES

[1] BUGYO, http://projects.celtic-initiative.org/bugyo

[2] http://www.cybertrust.com/

[3] BUGYO's Poster, http://projects.celtic-initiative.org/bugyo/our_vision2.htm

[4] CC. http://www.commoncriteriaportal.org/

[5] FIPA, http://www.fipa.org/

[6] JADE, http://jade.tilab.com/

[7] Aglet, http://aglets.sourceforge.net

[8] Tryllian, http://www.tryllian.org/

[9] JTXA, http://www.jxta.org

[10] OVAL, http://oval.mitre.org/

[11] Nagios, http://nagios.org/

[12] Java, http://java.sun.com/

[13] Developpez.com, http://www.developpez.com/

[14] http://www.skyboxsecurity.com/products/assure.html

[15] http://www.intellitactics.com/

APPENDICES

8.1 Detailed description of implemented methods

8.1.1 BUGYOAGENT

Name: *addResult*

Description: *This method add an XML element (measurement element) containing the value of the measurement (which is a string), the name of the target, … to the global result XML element (measurements element).*

Parameters:

[1] Name: *id* Type: *int*
Description: *the id of the measurement*

[2] Name: *value* Type: *String*
Description: *the value of the measurement*

[3] Name: *name* Type: *String*
Description: *the name of the target (full name or IP)*

[4] Name: *logs* Type: *String*
Description: *the logs returned by the probe during the measurement*

Return value: *none*

For this function we use the overloading method of java to declare three different functions with the same name, but a different signature. The difference between each function is the type of parameter value (String, Int, Float).

Name: *getServices*

Description: *This method returns measurements that the agent is able to manage.*

Parameters: *none*

Return value:

Type : *int[]*
Description : *The measurement's IDs that the agent is able to manage*

Name: *isUnix*

Description: *This method returns true if the running Operating System is an UNIX system.*

Parameters: *none*

Return value:

Type : *boolean*
Description : *True if the OS is UNIX, False else.*

Name: *isWindows*

Description: *This method returns true if the running Operating System is a Windows system.*

Parameters: *none*

Return value:

Type : boolean
Description : True if the OS is Windows, False else.

Name: *setup*

Description: *This method extends the setup function contained in the jade.core.Agent class. This method is used for startup tasks.*

Parameters: none

Return value: *none*

Name: *registerServices*

Description: *This method is used to declare measurement that the agent is able to perform.*

Parameters:

[1] Name: *myServices* Type: *int[]*
Description: *A table containing the measurement's IDs that the agent is able to manage. Each measurement is identified by its unique identifier*.

Return value: *none*

Name: *run*

Description: *This method is used to launch a probe via a command line.*

Parameters:

[1] Name: cmd Type: String
Description: The command line to execute
[2] Name : dir Type : String
Description: The directory in which the command has to be launched.

Return value:

Type : int
Description : The exit value of the Process launched. (0:OK)

Name: *run*

Description: *This method is used to launch a probe via a command line.*

Parameters:

[1] Name: cmd Type: String
Description: The command line to execute. By default, the command is launched in the directory containing the java calling class.

Return value:

Type : int

Description : The exit value of the Process launched. (0:OK)

Name: *runCommand*

Description: *This method is used to launch a probe via a command line for retrieving a measurement. The command line is red in an XML file called .xml and placed in the directory containing the agent class.*

Parameters:

[1] Name: id Type: int

Description: The measurement's identifier.

Name : target Type : String

The target's address. (IP or full name)

Return value: *none*

Name: *setXML*

Description: *This method is used transmit to the agent the measurements element.*

Parameters:

[1] Name: measurements Type: Element

Description: the measurements element containing the empty measurement elements.

Return value: *none*

Name: *takedown*

Description: *This method extends the setup function contained in the jade.core.Agent class. This method is used for cleanup tasks.*

Parameters: *none*

Return value: *none*

8.1.2 BUGYOINTERFACE

Name: *executeMeasurement*

Description: *This method is called for each measurement when the probe agent reads the XML element (metric). This method has to be implemented for each specific agent.*

Parameters: none

[1] Name: measurement Type: int
Description: the id of the measurement

[2] Name : target Type : String
The name of the target for which the measurement is required. It could be the name of the target, or its IP.

Return value: *none*

Name: *executeMeasurement*

Description: *This method is called when the probe agent reads the XML element (metric). This method has to be implemented for each specific agent.*

Parameters: none

[1] Name: measurements Type: int[]
Description: the ids of the measurement to measure

[2] Name : target Type : String
The name of the target for which the measurement is required. It could be the name of the target, or its IP.

Return value: *none*

8.1.3 APPLIANCEAGENT

Name: *dispatchXML*

Description: *This method reads an element and creates sub elements, which will be sent to the Bugyo Agent*

Parameters:

[1] Name: element Type: Element

Description: element representing the XML file received from Bugyo Server

Return value: *none*

Name: *addResultElement*

Description: *This method add an sub element to the result element, which will be sent to Bugyo when completed*

Parameters:

[1] Name: element Type: Element
Description: element representing the XML file received from Bugyo Agent

Return value: *none*

8.1.4 SERVERAGENT

Name: *setMeasuremts*

Description: *This method reads and parses an XML file to an element*

Parameters:

[1] Name: filename Type: String
Description: File's name containing the measurements to perform

Return value: *none*

Name: *saveElement*

Description: *This methods save an element to a file.*

Parameters:

[1] Name: filename Type: String
Description: File's name to save the measurement's result

Return value: *none*

Name: *sendElement*

Description: *This method sets the three variables passed as parameter and calls the RequestPerformer Behaviour.*

Parameters:

[1] Name: domainToCheck Type: String
Description: Appliance's sub-domain receiving the element

[2] Name: inputFilename Type: String
Description: File's name containing the measurements to perform

[3] Name: outputFilename Type: String
Description: File's name to save the measurement's result

Return value: *none*

8.2 Defining own definition file

Metadata

The first step is to write a Metadata that contains:

- OVAL-ID
- CVE-ID
- Status
- Platforms
- Date
- Schema version
- Description
- Definition synopsis

The metadata can look like this for example:

OVAL-ID: OVAL19	Date Modified: 2005-10-12
Status: ACCEPTED	Description
CVE-ID: CVE-2002-0189	Cross-site scripting vulnerability in Internet Explorer 6.0 allows remote attackers to execute scripts in the Local Computer zone via a URL that exploits a local HTML resource file, aka the "Cross-Site Scripting in Local HTML Resource" vulnerability.
Schema Version: 4	

Definition Synopsis: -- Vulnerable software exists

- the version of mshtml.dll is less than 6.0.2716.2200
- NOT the patch q321232 is installed (Installed Components key)
- NOT the patch q323759 is installed (Installed Components key)
- NOT the patch q328970 is installed (Installed Components key)
- NOT the patch q324929 is installed (Installed Components key)
- NOT the patch q810847 is installed (Installed Components key)
- NOT the patch q813489 is installed (Installed Components key)
- NOT the patch q818529 is installed (Installed Components key)
- NOT the patch q822925 is installed (Installed Components key)
- NOT the patch q828750 is installed (Installed Components key)
- NOT the patch q824145 is installed (Installed Components key)
- NOT Windows 2000 Service Pack 4 (or later) is installed
- Internet Explorer 6 is installed

-- Vulnerable configuration

For simplify the example we will take only the two first criteria:

- The version of mshtml.dll is less than 6.0.2716.2200
- NOT the patch q321232 is installed (Installed component key)

If these two criteria are true, then your system is vulnerable.

Pseudo Code

The second step is to write the pseudo code corresponding to the metadata. The pseudo code is a way of describing an algorithm without reference to a programming language in particular. Its descriptive aspect makes it possible to describe with more or less detail the algorithm, allowing of this fact of starting with a very broad vision.

Your machine is vulnerable to OVAL19 if ...

Vulnerable software section:

the version of mshtml.dll is less than 6.0.2716.2200

-- file_test:

- the file's path 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot\system32\mshtml.dll' exists
- the version is less than '6.0.2716.2200'

AND

(NOT) the patch q321232 is installed (Installed Components key)

-- registry_test:

- the hive 'HKEY_LOCAL_MACHINE' exists
- the key 'SOFTWARE\Microsoft\Active Setup\Installed Components\{D7B44F3E-77D3-44C5-8E03-4222D9A18B7B}' exists
- the name 'IsInstalled' exists
- the value equals 1

XML File

In the third step we have to write your definition file (see appendix).

```
<?xml version="1.0" encoding="UTF-8"?>
<oval xmlns="http://oval.mitre.org/XMLSchema/oval"
      xmlns:oval="http://oval.mitre.org/XMLSchema/oval"
      xmlns:windows="http://oval.mitre.org/XMLSchema/oval#windows"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval oval-schema.xsd ">
  <generator>
        <schema_version>4.2</schema_version>
        <timestamp>20060222204908</timestamp>
  </generator>
  <definitions>
        <definition id="OVAL19" class="vulnerability">
              <affected family="windows">
                    <windows:platform>Microsoft Windows 2000</windows:platform>
                    <product>Internet Explorer 6.0</product>
              </affected>
              <dates>
                    <submitted date="2004-01-27-05:00">
                          <contributor organization="The MITRE Corporation">Andrew
                          Buttner</contributor>
                    </submitted>
                    <modified date="2005-09-20-04:00" comment="Changed IE registry test to
                    wrt-18">
                          <contributor organization="The MITRE Corporation">Christine
                          Walzer</contributor>
                    </modified>
                    <status_change date="2005-09-21-01:27">INTERIM</status_change>
                    <status_change date="2005-10-12-05:49">ACCEPTED</status_change>
              </dates>
              <description>
                    Cross-site scripting vulnerability in Internet Explorer 6.0 allows remote
                    attackers to execute scripts in the Local Computer zone via a URL that exploits
                    a local HTML resource file, aka the "Cross-Site Scripting in Local HTML
                    Resource" vulnerability.
              </description>
              <reference source="CVE">CVE-2002-0189</reference>
              <status>ACCEPTED</status>
              <version>4</version>

              <criteria>
```

```
                    <software operation="AND">
                        <criterion test_ref="wft-204" comment="the version of mshtml.dll is
                        less than 6.0.2716.2200" negate="false" />
                        <criterion test_ref="wrt-204" comment="the patch q321232 is
                        installed (Installed Components key)" negate="true" />
                    </software>
                </criteria>
            </definition>
        </definitions>
        <tests>
            <registry_test id="wrt-204" comment="the patch q321232 is installed (Installed Components
            key)" check="at least one" xmlns="http://oval.mitre.org/XMLSchema/oval#windows">
                <object>
                    <hive>HKEY_LOCAL_MACHINE</hive>
                    <key>SOFTWARE\Microsoft\Active Setup\Installed Components\{D7B44F3E-
                    77D3-44C5-8E03-4222D9A18B7B}</key>
                    <name>IsInstalled</name>
                </object>
                <data operation="AND">
                    <value datatype="int" operator="equals">1</value>
                </data>
            </registry_test>

            <file_test id="wft-204" comment="the version of mshtml.dll is less than
            6.0.2716.2200" check="at least
            one" xmlns="http://oval.mitre.org/XMLSchema/oval#windows">
                <object>
                    <path datatype="component">
                        <component type="registry_value">HKEY_LOCAL_MACHINE\SOFTW
                        ARE\Microsoft\Windows
                        NT\CurrentVersion\SystemRoot</component>
                        <component type="literal">\system32\mshtml.dll</component>
                    </path>
                </object>
                <data operation="AND">
                    <version datatype="version" operator="less than">
                        <major>6</major>
                        <minor>0</minor>
                        <build>2716</build>
                        <private>2200</private>
                    </version>
                </data>
            </file_test>
```

```
        </tests>
</oval>
```

8.3 Using OVAL Definition Interpreter

Required Privileges

In order to collect all of the system configuration data required to correctly evaluating OVAL Definitions, the Definition Interpreter MUST BE RUN WITH ADMINISTRATOR/ROOT PRIVILEGES.

Certain system data referenced by OVAL Definitions is only available to privileged accounts. This includes information about running processes, and potentially registry key and file information (depending on local security settings). While it is possible to run the Definition Interpreter as a non-privileged user, the results of the analysis may not convey the true state of the system.

Data Protection

The Definition Interpreter collects system configuration data only available to a user with Administrator/root access. That data may be stored locally in a XML file. In addition, the vulnerability status of the system is written to a file. SINCE THIS IS SENSITIVE INFORMATION, IT IS STRONGLY RECOMMENDED THAT THE DEFINITION INTERPRETER DIRECTORY BE RESTRICTED TO ADMINISTRATOR ACCESS ONLY.

Quick Usage Guide:

1) As the Interpreter is only a reference implementation, it has purposefully been designed as solely a command-line utility. Therefore, to execute the Interpreter, you will first have to open a command window, and change to the Interpreter installation directory.

2) If you have downloaded a newer data file than the one bundled with the Interpreter, move the new data file into the Interpreter installation directory, and rename it to 'definitions.xml'.

3) Run the Interpreter, supplying the MD5 checksum of the data file as a command line option.

> ovaldi MD5Hash

After verifying the integrity of the data file using the MD5 checksum, the XML Definition Interpreter will read the 'definitions.xml' file to determine what data to collect from the system, will analyze the collected data against the 'definitions.xml' file, and will report its findings.

Advanced Usage:

The Definition Interpreter accepts a number of command-line options.

Command Line: ovaldi [options] MD5Hash

Here are most important:

-m -- Run without requiring an MD5 checksum. Running the Interpreter with this option DISABLES an important security feature. In normal usage, a trusted checksum provided on the command line is used to verify the integrity of the OVAL Definitions file.

-o -- Specifies the pathname of the OVAL Definition file to use. If none is specified than the Interpreter will default to "definitions.xml" in the Interpreter directory.

-z -- Calculates and prints to the screen the MD5 checksum of the current data file

MD5Hash -- The MD5 checksum expected for the current data file (definitions.xml by default, or as specified by the -o option). The Interpreter calculates the actual checksum of the Data file, and compares it to this value provided on the command line, to verify the data file's integrity. Checksums are available from the OVAL Web site.

The checksum verification ensures that the data file has not been modified: that the OVAL definitions have not been tampered with, or potentially malicious content added. Unless the -m option is specified, the MD5Hash is REQUIRED to run the Interpreter.

Interpreter's installation

Then we have to download the OVAL Definition Interpreter from OVAL website:

http://oval.mitre.org/oval/download/interpreter/index.html

The Windows download is a self-extracting archive (.exe) and the Red Hat download is an RPM (.rpm) file.

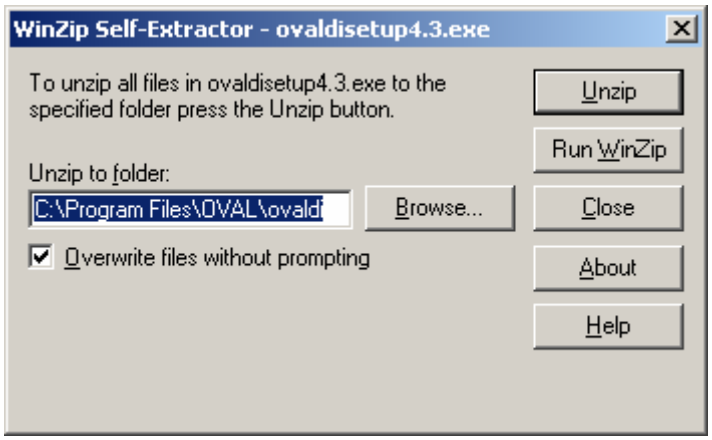

Figure 28 - Windows self-extracting archive

After installing the Interpreter, we will copy the last version of definition file in the same directory.

Running Interpreter

Therefore, to execute the Interpreter, you will first have to open a command window, and change to the Interpreter installation directory (e.g C:\Progams Files\OVAL\ovaldi). Then launch the Interpreter with this command:

- C:\OVAL\ovaldi>ovaldi.exe –m

The option –m is very important, if we omit it, the Interpreter will not execute. This option enables to run the Interpreter without requiring an MD5 checksum.

C:\WINNT\system32\cmd.exe
C:\OVAL\ovaldi>ovaldi.exe -m
OVAL XML Definition Interpreter
Version: 4.3 Build: 12
Build date: Oct 12 2005 09:14:53
Copyright (c) 2004 - The MITRE Corporation
Tue Mar 07 10:49:15 2006
** parsing definitions.xml file.
- validating xml schema.
** checking schema version
- Definition Interpreter version - 4.3
- Schema version - 4.2
** parsing mapping.xml
- validating xml schema.
** removing unneeded content from definitions.xml
** creating a new data document.
** gathering data for the OVAL definitions.
** saving data model to system-characteristics.xml.
** running the data analysis.
- This may take a while.
Performing analysis on: citi-8203.private.tudor.lu
** OVAL definition results.
VULNERABILITIES FOUND:
OVAL Id Reference Source Reference Id
None.
VULNERABILITIES NOT FOUND:
OOVAL Id Reference Source Reference Id
OVAL19 CVE CVE-2002-0189
UNABLE TO DETERMINE RESULT:
OVAL Id Reference Source Reference Id
None.
** finished evaluating OVAL definitions.
** saving detailed results to results.xml.

Figure 29 - Execution of the Interpreter

For each OVAL Definitions file, characteristics of the tested system are searched. The OVAL System Characteristics is created.

Then characteristics of the tested system and characteristics needed for the vulnerability are compared. If they are the same, it means the system is vulnerable.

Result File

Finally, a report containing all vulnerabilities discovered on the tested system is created. The OVAL Result file is created.

By adding a reference to an xsl file, we can visualize the result file in Internet Explorer.

```
<?xml:stylesheet type="text/xsl" href="results_to_html.xsl"?>
```

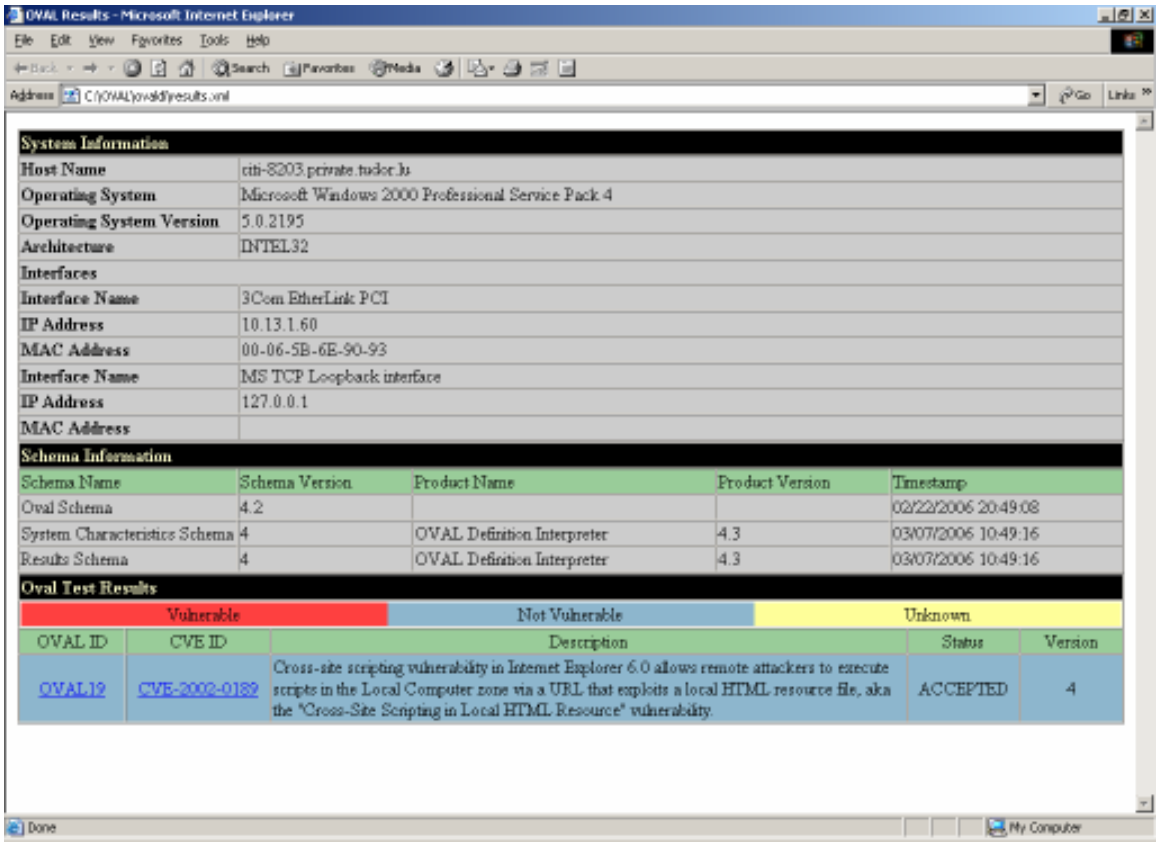
OVAL Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\OVAL\ovaldi\results.xml
System Information
Host Name citi-8203.private.tudor.lu
Operating System Microsoft Windows 2000 Professional Service Pack 4
Operating System Version 5.0.2195
Architecture INTEL32
Interfaces
Interface Name 3Com EtherLink PCI
IP Address 10.13.1.60
MAC Address 00-06-5B-6E-90-93
Interface Name MS TCP Loopback interface
IP Address 127.0.0.1
MAC Address
Schema Information
Schema Name Schema Version Product Name Product Version Timestamp
Oval Schema 4.2 02/22/2006 20:49:08
System Characteristics Schema 4 OVAL Definition Interpreter 4.3 03/07/2006 10:49:16
Results Schema 4 OVAL Definition Interpreter 4.3 03/07/2006 10:49:16
Oval Test Results
Vulnerable
Not Vulnerable
Unknown
OVAL ID CVE ID Description Status Version
OVAL19 CVE-2002-0189 Cross-site scripting vulnerability in Internet Explorer 6.0 allows remote attackers to execute scripts in the Local Computer zone via a URL that exploits a local HTML resource file, aka the "Cross-Site Scripting in Local HTML Resource" vulnerability. ACCEPTED 4
Done
My Computer

Figure 30 - OVAL results

8.4 Install and run Nagios

This chapter presents a short tutorial how to install and run Nagios on SUSE Linux 10.1

1. Install SUSE Linux
2. Install lampp (see appendix X)
3. Download every Nagios RPMs for SUSE 10.1

http://www.nagiosexchange.org/SuSE_Linux.74.0.html?&tx_netnagext_pi1%5Bp_view%5D=569

4. Save the RPMs on /home/{user}/nagios

5. As root, install these RPMs:

   Changing to root: localhost$ su - [ENTER]

   [root password]

   Got to the directory /home/{user}/nagios:

   localhost# cd /home/{user}/nagios

   Install the RPMs:

   localhost# rpm -ihv nagios*rpm

   This install every RPM located in the directory and which corresponds to the pattern. Normally, Nagios is yet installed.

6. Add Nagios to runlevels 3 (multiuser+network) and 5 (multiuser+network+GUI):
   - Launch yast: K Menu - System - Yast
   - On System tab, click on System Services (Runlevel)
   - Go to Expert Mode and find Nagios service
   - Select runlevel 3 and 5
   - Validate

   Normally, Nagios should be started automatically at runlevel 3 and 5

7. Launch Nagios

   By restart the system

   By reinitializing the runlevel:

   As root: localhost# init 3

   localhost# init 5

   By launching Nagios service:

   As root:

   localhost# service nagios start

8. Link http://localhost/nagios to the directory containing the web interface of:

   Open file /opt/lampp/etc/httpd.conf

   Add these lines (at the end):

   #NAGIOS

   Include etc/extra/httpd-nagios.conf

9. Create file /opt/lampp/etc/extra/httpd-nagios.conf

10. Insert in this file (between the stars):

```
**************************************
ScriptAlias /nagios/cgi-bin /usr/lib/nagios/cgi

<Directory "/usr/lib/nagios/cgi">
        Options ExecCGI
        AllowOverride None
        Order allow,deny
        Allow from all
        AuthName "Nagios Access"
        AuthType Basic
        AuthUserFile /etc/nagios/htpasswd
        Require valid-user
</Directory>
```

```
Alias /nagios /usr/share/nagios

<Directory "/usr/share/nagios">
        Options None
        AllowOverride None
        Order allow,deny
        Allow from all
        AuthName "Nagios Access"
        AuthType Basic
        AuthUserFile /etc/nagios/htpasswd
        Require valid-user
</Directory>
****************************************
```

N.B.: These lines are not the same as these indicated on the web site http://www.nagios.org, because the files on SUSE Linux 10.1 are not installed in the same directory.

The directory containing the CGI-BIN on SUSE Linux 10.1 is: /usr/lib/nagios/cgi

The directory containing the web interface is: /usr/share/nagios

The file containing the AuthUserFile is: /etc/nagios/htpasswd

Login/password for web site:

Login: nagadmin

Password: nagios

11. Save this file
12. Restart lampp:

    localhost# service lampp restart [ENTER]

13. Web interface can be accessed by: http://localhost/nagios

8.5 Install XAMPP 1.5.2 on SUSE Linux 10.1

1. Download .tar.gz XAMPP 1.5.2 file for Linux on http://www.apachefriends.org to directory /home/{user}

2. Install XAMPP:

```
localhost$ su - [ENTER]
           [enter password]
localhost# cd /home/{user}
localhost# tar xzvf xampp-linux-1.5.2.tar.gz -C /opt
```

Normally, XAMPP is installed on /opt/lampp

4. Link the start script of lampp to directory /etc/init.d

   localhost# ln -s /opt/lampp/lampp /etc/init.d/lamp

   Without this command, Yast won't be able to find the lamp service

4. Add lampp to runlevels 3 and 5:

   Open Yast, System - System Services (Runlevels):

   On expert mode select lamp service

   Check runlevel 3 and 5

   Validate

5. Launch lamp

   By restart the system

   By reinitializing the runlevel:

   As root: localhost# init 3

   localhost# init 5

   By launching Nagios service:

   As root:

   localhost# service lampp start

6. Configure login/password

   As root: localhost# /opt/lampp/lampp security [ENTER]

   Enter login and password

Lampp should now be correct installed (MySQL 5, PHP 5.1):

- http://localhost/
- http://localhost/phpmyadmin

GLOSSARY

Term	Description
ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
API	Application Programming Interface
ASDK	Aglet Software Development Kit
ATP	Agent Transport Protocol
BADK	BUGYO Agent Development Kit
CC	Common Criteria
DF	Directory Facilitator
FIFO	First In First Out
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphic User Interface
JDK	Java Development Kit
JRE	Java Runtime Environment
JSP	Java Server Pages
KQML	Knowledge Communication Meta Language
LEAP	Lightweight Extensible Agent Platform
MAL	Metric Assurance Level
MAS	Multi Agent System
MASIF	Mobile Agent System Interoperability Facilities
MTP	Message Transport Protocol
MVC	Model-view-controller
NE	Network Entity
NMS	Network Management Systems
OSS	Operation Support Systems
RMI	Remote Method Invocation
RTT	Round Trip Time
SL	Semantic Language
SSL	Secure Sockets Layers
TLS	Transport Layer Security